Une modélisation de la variabilité multidimensionnelle pour une évolution incrémentale des lignes de produits

Stephen Creff

N^o d'ordre : XXXX Année 2013

THÈSE / UNIVERSITÉ DE RENNES 1

sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

Stephen Creff

préparée à l'unité de recherche UMR 6074 IRISA

Équipes d'accueil : ENSTA Bretagne STIC/IDM, LabSTICC - IRISA/Triskell

Convention CIFRE N^o : 0004/2009

Une modélisation de la variabilité multidimensionnelle pour une évolution incrémentale des *lignes de produits*

Thèse soutenue à Brest

le 09 décembre 2013

devant le jury composé de :

Antoine BEUGNARD
Professeur à Télécom Bretagne / président

Philippe COLLET
Professeur à l'Université de Nice Sophia-Antipolis / rapporteur

Pierre-Alain MULLER
Professeur à l'Université de Haute-Alsace / rapporteur

Arnaud MONÉGIER du SORBIER
Architecte Logiciel à Thales Air Systems / examinateur

Jean-Marc JÉZÉQUEL
Professeur à l'Université de Rennes 1 / directeur de thèse

Joël CHAMPEAU
Enseignant-chercheur à l'ENSTA Bretagne / co-directeur de thèse

Une modélisation de la variabilité multidimensionnelle pour une évolution incrémentale des *lignes de produits*

Dissertation

pour l'obtention du grade de
DOCTEUR DE L'UNIVERSITÉ DE RENNES 1,
Mention : Informatique,

défendu publiquement le 09 12 2013,
par,

Stephen Creff

"I KEEP six honest serving-men
(They taught me all I knew);
Their names are What and Why and When
And How and Where and Who."
— Rudyard Kipling, *The Elephant's Child (1902)*

Remerciements

Tout au long de ce travail, et plus généralement de mon parcours réalisé ces dernières années, j'ai pu bénéficier de soutiens, de conseils et d'encouragements d'un très grand nombre de personnes auxquelles je tiens à exprimer toute ma gratitude.

Je tiens tout d'abord à remercier Arnaud Monégier du Sorbier, pour avoir initié cette thèse CIFRE à Thales Air Systems, et sans qui cette épisode n'aurait put être envisageable.

Merci à Joël Champeau, mon encadrant dans l'équipe d'accueil, et à Jean-Marc Jézéquel, mon directeur de thèse, de m'avoir apporté un soutient dans la conduite de mes travaux.

Merci à Philippe Collet et Pierre Alain Muller d'avoir accepté d'évaluer ce mémoire, ainsi qu'à Antoine Beugnard d'avoir consenti à présider le jury de soutenance.

Ces années de doctorat est de prime abord une aventure professionnelle et d'apprentissage de la recherche, mais également une aventure humaine. Merci à tous mes collègues de l'ENSTA Bretagne (ex. ENSIETA ...) à Brest, de Thales Air Systems à Rungis, de l'équipe Triskell de l'IRISA à Rennes, et aux personnes gravitantes autour de ces entités pour les discussions et pour tout ce qu'ils m'ont apporté (atmosphère de travail, points de vue métier, ...), mes travaux se sont nourris de ces échanges.

Enfin, et non des moindres, un grand merci à ma famille et mes amis pour leur soutient, leur aide, ainsi que leur présence inestimée à mes côtés durant toutes ces années.

Sommaire

CHAPITRE 1

Introduction

"If we knew what it was we were doing, it would not be called research, would it ?"

—Albert Einstein

Avant-propos

Le doctorat s'inscrit dans le cadre d'une bourse CIFRE, entre les univers académique et industriel. Les préoccupations de Thales Air Systems, et plus particulièrement de l'équipe de rattachement, sont les systèmes de défense anti-aériens qualifiés de systèmes à logiciels prépondérants. La section 1.1, contextualisante, montre l'importance du logiciel dans notre société actuelle et met, dans un premier chapitre, l'accent sur le contexte du génie logiciel, son orientation vers l'Ingénierie Dirigée par les Modèles (IDM) et les *lignes de produits* (LdPs). La section 1.2 pose la problématique et identifie certains besoins liés à l'évolution d'une Ligne de Produits de modèles. La section 1.3 présente les deux volets constituant la contribution de la thèse, se focalisant dans un premier temps sur l'ingénierie du domaine puis dans un second temps sur celle de l'application. Enfin, la dernière section détaille le plan du document de thèse.

1.1 Contexte

La société actuelle côtoie de manière familière le monde numérique, dans lequel l'ordinateur devient anecdotiquement représenté. L'omniprésence des systèmes logiciels dans notre quotidien est aussi diverse que variée, allant des systèmes spécifiques domotiques aux *smartphones* qui envahissent majoritairement le marché. Les systèmes de défense ne font pas exception et les logiciels embarqués critiques y sont très répandus. L'ingénierie des systèmes de défense (avionique, systèmes de contrôle anti-aériens, radars) est soumise à des contraintes fortes au regard de la sûreté de fonctionnement, de la sécurité ou des performances pour ne citer que ces exemples.

L'ingénierie du génie logiciel apparaît à la fin des années 1960 en réponse à la complexité croissante des systèmes logiciels, due à une demande toujours croissante de qualité, de nombre de fonctionnalités attendues et de volume de données à traiter, ainsi qu'à l'hétérogénéité des plateformes et des technologies impliquées, qui transparaît dans son architecture. À titre d'exemple,

il apparaît normal et même banal, à tout à chacun, de pouvoir visionner une vidéo en temps réel sur son téléphone. Le génie logiciel se penche sur les artefacts produits et également sur les processus de réalisation de ces derniers.

1.1.1 Ingénierie Dirigée par les Modèles

Pour relever le défi de la complexité croissante, la communauté du génie logiciel voit émerger, il y a quelques années, l'Ingénierie Dirigée par les Modèles [CESW04, FR07]. Cette dernière apparaît également sous la dénomination de Développement Dirigé par les Modèles ou Architecture Dirigée par les Modèles. Les expressions dans la langue anglaise sont respectivement, et dans l'ordre d'apparition chronologique, *Model-Driven Architecture (MDA)*, *Model-Driven Development (MDD)*, *Model-Driven Engineering (MDE)* et *Model-Based Engineering (MBE)*. L'IDM fait la part belle à l'abstraction et à la séparation des préoccupations pour tenter de canaliser, résorber et maîtriser la complexité intrinsèque des systèmes à développer. L'abstraction est un concept qui concentre l'intégralité du fondement des questions portant sur la nature de la pensée. L'IDM inscrit l'artefact de modélisation comme base conceptuelle à l'aide de laquelle toute activité de raisonnement se déploie. Les modèles sont une représentation simplifiée d'un aspect de la réalité pour un objectif donné. Ces modèles, qui ont une substituabilité bornée, sont usuellement analytiques et la pensée du concepteur est, conséquemment, à distance de ce sur quoi elle porte. Les raisonnements et analyses sont réalisés dans des espaces technologiques dédiés [Fav05a], offrant un cadre méthodologique et outillé adapté.

L'IDM essaie de lire le complexe réel sous l'apparence "simple" fournie par les modèles et apporte une approche pragmatique permettant d'augmenter le niveau d'abstraction des développements. Les concepteurs se focalisent sur des artefacts dans un spectre réduit de préoccupations à l'aide de langages spécifiques à leur domaine *(Domain Specifique Languages - DSLs)* et de points de vue (encapsulation d'informations ciblées sur le métier de l'utilisateur). La légitimité des points de vue tient, évidemment, à leur intelligibilité et à leur communicabilité. Il faut en voir l'utilité dans un contexte très général : un modèle à un niveau d'abstraction donné peut être abordé de plusieurs points de vue transverses et spécifiques à un domaine métier, selon que l'observateur considère, par exemple, ses propriétés de sûreté de fonctionnement, de performance, de persistance, de sécurité ou de tolérance aux pannes.

1.1.2 Ligne de Produits

La compétitivité requise par le marché, dans le domaine grand public comme dans ceux de l'aéronautique, des transports et de la défense, impose aux industriels des contraintes de plus en plus drastiques de minimisation des coûts, ainsi que des délais de développement et de mise sur le marché. À cela s'ajoutent des demandes de qualité des produits, de sûreté et de sécurité toujours accrues. Une direction suivie par la communauté des *lignes de produits* consiste à capitaliser sur le savoir-faire métier dans un domaine spécifique, tant sur le processus que sur les objets produits eux-mêmes.

La philosophie de réutilisation du logiciel sous forme de *lignes de produits* consiste en la réutilisation intra-organisationnelle réalisée grâce à l'exploitation explicitement planifiée de similitudes entre les produits connexes. Une LdP est une famille de produits logiciels qui sont construits de manière prescrite, à partir d'un ensemble commun de ressources de développement de base [PBL05].

La gestion de la variabilité est un élément clé dans l'ingénierie des *lignes de produits*. Plusieurs techniques existent pour identifier et expliciter les points de variations marquant les divergences entre les produits d'une même gamme. Une technique, originellement présentée en 1990, et de-

puis largement répandue à travers la communauté LdP, est l'utilisation de *"Feature Diagrams"* [KCH+90, BFG+02] (parfois nommés diagrammes de caractéristiques en français). Ces modèles révèlent les points de variation et relient les divergences par des opérateurs logiques tels que par exemple : *ET*, *OU*, *OU exclusif*, et des contraintes d'exclusion mutuelle et d'implication plus complexes. D'autres langages dédiés peuvent également être employés à l'exemple de *OVM* [PBL05].

Les approches modernes de développement logiciel par la modélisation offrent de grandes opportunités pour comprendre et gérer les éléments communs et variables d'une *ligne de produits* en se basant sur des perspectives différentes. Les *Model Driven Software Product Lines (MD-SPLs)* [CAK+05], ou *ligne de produits dirigées par les modèles*, tendent à réaliser une synergie de l'abstraction de l'IDM et de la capacité de gestion de la variabilité des LdPs.

1.1.3 En résumé

Les systèmes embarqués critiques, tels ceux du monde de la défense, sont complexes et coûteux à réaliser. Ils possèdent, en outre, une part, plus ou moins importante de similitudes. Des gammes de produits sont présentes et des points communs sont identifiables entre différentes affaires espacées temporellement. Les produits sont des logiciels embarqués dont le développement requiert l'application de fortes contraintes architecturales. Dans ce contexte, le processus classique de développement posant un nivellement systémique (système, logiciel, et matériel) et technologique (espace du problème, solution logique, et solution physique) doit être combiné aux processus de réutilisation des *lignes de produits* (Ingénierie du domaine et de l'application).

1.2 Problématique de l'évolution des *lignes de produits*

À l'instar des logiciels plus "traditionnels", les *lignes de produits* à logiciel prépondérant sont sujettes au vieillissement - phénomène observé et décrit par David Parnas [Par94] - et suivent entre autres les lois de l'évolution du logiciel postulées par Lehman [LR01], mais pas uniquement.

Les *lignes de produits*, dans leur principe premier, se donnent pour objet de prévoir un nombre limité de configurations connues *a priori*, constituant une *famille de produits*, permettant de produire une gamme d'entités. Cette approche est le plus souvent proactive et non réactive, à l'exemple de la *succes story* de Nokia dans la téléphonie mobile [LSR07]. Dans le cas d'une production de masse, du type téléphone ou automobile, les configurations et options possibles sont prévues à l'initialisation et une évolution de ces lignes de produits entraîne généralement une refonte de la conception de la famille.

La réalité se présente en des termes bien dissuasifs, notamment dans le cadre de LdPs de systèmes militaires, tant la part d'imprévu est grande tout au long du cycle de vie de la famille. Le client est force de proposition et une ligne de produits proactive avec un configurateur préétabli ne peut être mise en application. Citons pour exemple d'imprévus : l'émergence de nouveaux besoins opérationnels, la naissance de nouvelles plate-formes technologiques ou, tout simplement, le besoin de retouches correctives sur certains éléments de la LdP. À une échelle de temps plus courte les concepteurs peuvent observer des glissements dans les spécifications dans la définition d'un nouveau produit.

La complexité initialement décrite dans le système résultant est également présente dans l'espace du problème et peut être perçue comme la caractéristique "d'un état de chose [...] qui nous égare, nous interdit la moindre prévision, nous ôte toute possibilité de raisonner sur l'avenir"[1].

[1] P. Valery, Regards sur le monde actuel, Œuvres, Pléiade, t.II, p. 1059

Nul n'est extralucide et il est utopique de penser pouvoir prévoir et planifier ce que seront les (futurs) besoins des (futurs) utilisateurs. Afin de limiter les risques en termes de surcoût engendré par des inspirations personnelles qui peuvent se révéler hasardeuses, une approche proactive est donc à proscrire dans le contexte de LdPs de systèmes embarqués.

En outre, la nature irrévocablement dynamique des systèmes réels induit une évolution permanente des LdPs afin de répondre aux nouvelles exigences des clients et pour refléter les changements des artefacts internes de la *ligne de produits*. La LdP doit être ouverte à l'évolution.

Toutefois, les approches dans la littérature ne proposent pas de cycle d'évolution type, et les expériences reportées font écho de coût de ré-ingénierie élevé. Une approche systématique de prise en considération de l'évolution est essentielle et elle ne passe pas forcement par de la planification exhaustive des cas possibles.

1.3 Motivation et Contribution

Force est de constater le manque de réactivité du processus des LdPs en présence d'une évolution nécessaire et irrémédiable de la famille de produits. Face à la complexité des systèmes de la LdP et au besoin non assouvi d'un processus d'évolution, les questions suivantes se posent :

- Comment abstraire de manière cohérente la variabilité de la famille de produits ?
- Quelle représentation et organisation des artefacts de modélisation est à fournir pour permettre une compréhension et un raisonnement efficace ?
- Le traitement de la variabilité existante est-il suffisant pour des systèmes embarqués à forte hétérogénéité et contraintes architecturales ?
- Comment organiser les artefacts de modélisation pour obtenir une aisance d'évolution ?
- Quel est le processus d'évolution envisagé/envisageable pour les systèmes réactifs de défense de Thales ?
- Dans une perspective d'évolution, quels efforts de développement peuvent être commonalisés ? Le(s) concepteur(s) peut (peuvent) il(s) être assisté(s) tout au long de ce processus ?

Ces interrogations peuvent être classées dans deux catégories, les quatre premières étant liées à la gestion de la variabilité dans les lignes de produits, les dernières étant liées à un processus d'évolution des LdPs. Ces deux aspects sont source des challenges exposés ci-après.

1.3.1 Le challenge de la séparation des préoccupations et de la variabilité

Parmi les approches actuelles de la communauté des LdPs, certaines font le lien entre les modèles de variabilité et des fragments de modèles de base [CA05, PBL05, PKGJ08, HKW08]. Ces approches se focalisent sur la variabilité et mettent à plat l'abstraction et la séparation des préoccupations mises en exergue par l'IDM, se ramenant à une gestion des LdPs qu'il est possible de qualifier d'historique, à savoir identique à une gestion variabilité/code des LdPs. Un diagramme de caractéristique est communément proposé dans l'espace du problème, mis en regard d'éléments de solutions techniques. Les approches existantes se basent sur un socle technologique de type intergiciel important, avec des contraintes de plate-forme figées, un cadre figé, ce qui n'est pas le cas dans nos systèmes. L'absence de représentation claire et distincte de l'hétérogénéité du problème et de la solution entraîne dans certain cas une complexité artificielle des relations entre les points de variation et les éléments techniques de la solution.

L'importance de la problématique de la modularité des modèles de variabilité a été soulignée par de nombreux auteurs [CHE04, BSP09, RW06, MPH+07, TTBC+09, ACLF10]. Cependant, la présente approche propose de renforcer la dichotomie entre une variabilité de l'espace du problème - propre au domaine du marketing et mis en regards de *Uses-cases*, *Goal-cases* et exigences - et une variabilité de l'espace de la solution - technique et propre à l'ingénierie, mise en regards de

modèles logiques et physiques. Cette thèse propose de tirer pleinement parti de l'*ingénierie des modèles* et avance une approche qui prône une décomposition de la variabilité, tout au long du processus de développement - au travers des niveaux d'abstraction contexte, logique et physique - prenant en considération des préoccupations métiers - par ex. sûreté de fonctionnement, sécurité, performance -, au travers de diverses ingénieries - système, logiciel et matériel (propos introduits dans [CMCJ11]), résultant en une variabilité multidimensionnelle.

Dans ce contexte, l'information relationnelle apparaît partout, par exemple dans l'identification de chevauchement d'information entre les modèles ou de leurs interactions [CNS11]. Par conséquent, il est essentiel pour les instances qui traitent des connaissances - que ce soit des êtres humains ou des systèmes de traitement de la connaissance - non seulement de classer conceptuellement les entités d'un domaine de discours comme des objets isolés, mais également de décrire les relations entre eux. Les approches de modélisation de la variabilité existante mettent en avant le besoin d'expression de diverses relations entre éléments de la LdP (propos introduits dans [CC12]). Cependant, les observations montrent que ces relations manquent fréquemment de formalisation, tant sur leurs contenus sémantiques, que sur leurs usages. La thèse défend que la compréhension des relations entre les différents objets utilisés dans le développement de logiciels est essentielle pour assurer que le système livré est conforme aux besoins des parties prenantes, d'où l'importance d'un raisonnement sur la nature intentionnelle des modèles [MFBC10] et d'une organisation de l'espace de modélisation [DGRN10]. La proposition repose sur un langage de modélisation dédié à l'expression relationnelle entre les artefacts et éléments d'artefacts d'une *ligne de produits dirigées par les modèles* (propos introduits dans [CCJM12b, CC13]). L'aspect relationnel est isolé dans le DSL de modélisation pour en renforcer la sémantique, et organisé en points de vues. Les relations opérationnelles dans les activités de la LdP, par ex. vérification de cohérence, dérivation, modélisation structurelle, spécialisent des relations intentionnelles de granularité supérieure (de type "*representation-of*" entre modèles [MFBC10]) ; ces dernières définissent un sous ensemble de l'intention d'un processus de développement donné [KC10].

1.3.2 Le challenge du processus flexible guidant l'évolution des LdPs

Le manque de processus type pour faire évoluer une *ligne de produits*, ainsi qu'un manque de flexibilité et d'automatisme pour cette évolution ont précédemment été mis en lumière. Des études sur l'évolution et la modification de modèles de variabilité existent dans la littérature [AGM+06] mais elles ne prennent pas en compte la globalité des préoccupations de la LdP, notamment l'architecture. Dans une approche réactive globale, l'objectif sur le plan financier est également d'avoir un investissement et un retour sur investissement (*return on investment - ROI*) par incréments, chaque incrément d'évolution fournissant des bénéfices suffisants. Une approche réactive ne requiert pas d'investissement démesuré, néanmoins les coûts de ré-ingénierie et de modernisation de la LdP peuvent être faramineux sans une prise en considération initiale de l'architecture [Bos00] et du processus. Sur la base de l'infrastructure présentée dans la section précédente, est construit un processus d'évolution incrémentale de la *ligne de produits*.

D'autre part, il est couramment admis qu'une spécialisation spécifique produit soit réalisée lors de la dérivation. Une équipe réalise pour un projet des développements spécifiques en marge de la *ligne de produits*. Des observations [DSB05] révèlent un manque de support méthodologique pour l'ingénierie de l'application et, par conséquence, les organisations échouent dans l'exploitation exhaustive des bénéfices de la LdP. Perrouin et al. [PKGJ08] observent, quant à eux, un manque de flexibilité de ce processus de dérivation et proposent une approche permettant une extension du périmètre de la LdP pour bénéficier au mieux de cette dernière. Le dessein de la thèse est d'aller plus loin et, par suite, de fournir un processus outillé pour faciliter la réinjection des développements dans la LdP et ainsi rentabiliser au mieux les efforts fournis : une évolution

incrémentale de la Ligne de Produit par extension (propos introduits dans [CCJM12a]). La proposition marque un rapprochement entre le processus de dérivation de l'ingénierie de l'application et la gestion de la variabilité propre à l'ingénierie du domaine. Ce faisant, des règles de cohérence architecturale sont abordées pour délimiter la portée de la famille de produit, contraindre le processus de dérivation et faciliter les futures évolutions (propos introduits dans [CCMJ12]).

1.3.3 Explicitation de l'objectif et des contributions de la thèse

L'objectif est unique : maîtriser des incréments d'évolution d'une Ligne de Produits de systèmes complexes. La thèse est que 1^o) une variabilité multidimensionnelle et une identification claire des préoccupations sont requises dans le cadre de *lignes de produits* de systèmes complexes pour en améliorer la compréhension et faciliter l'évolution et que 2^o) les efforts de spécialisation lors de la dérivation d'un produit doivent être capitalisés dans un processus semi-automatisé d'évolution incrémentale des *lignes de produits*.

Pour résumer, deux points ressortent de cette thèse :

1. D'une part, améliorer la représentation et la compréhension du savoir et donc des artefacts de modélisation dans un processus d'ingénierie complexe de *ligne de produits* de systèmes embarqués. L'application de la séparation des préoccupations dans des points de vue sera la base de l'organisation de l'espace de modélisation. Ce faisant, des règles de cohérence architecturale seront abordées pour délimiter la portée de la famille de produit, contraindre le processus de dérivation et faciliter les futures évolutions.
2. D'autre part, proposer une évolution incrémentale de la Ligne de Produit par extension. L'objectif est ici d'apporter une flexibilité et une aide dans le processus d'évolution des LdPs en bénéficiant des efforts réalisés lors de la phase de dérivation des produits.

Les contributions pour y parvenir sont multiples :

C 0 : *Une étude des relations et de leur sémantique et usages dans les approches des lignes de produits.*

C 1 : *Un cadre conceptuel générique de décomposition multidimensionnelle de la variabilité suivant quatre axes.*

Ce cadre intègre une approche systémique de la définition de l'espace de modélisation de la *ligne de produits* : application du motif contexte - problème - solution, suivant quatre dimensions (processus, abstraction de modélisation, ingénierie, et préoccupation métier).

C 2 : *Une modélisation explicite de l'intentionnalité de l'espace de modélisation des Lignes de Produit.*

C 3 : *Une modélisation des interdépendances et relations entre les éléments de modèles, et de la sémantique liée à leurs usages.*

Ces deux contributions constituent les deux volets de la définition d'un langage de modélisation (*Domain Specific Modeling Language - DSML*) nommé PLiMoS permettant une modélisation centrée sur l'aspect relationnel de l'espace de modélisation des *lignes de produits*. Par le biais de ces deux contributions est mise en exergue l'importance des relations et de leur modélisation dans les processus de développements logiciel.

C 4 : *Une structuration de l'espace de modélisation par la définition d'une plate-forme conceptuelle basée sur un patron pour guider et contraindre la dérivation et l'évolution de la Ligne de Produits.*

La contribution permet de raffiner la définition du périmètre de la *ligne de produits* et de cibler de son évolution. Le cadre applicatif introduit un patron nommé ABCDE, adapté aux systèmes réactifs des applications de défense de Thales Air Systems.

C 5 : *Un processus d'évolution incrémental des lignes de produit par extension qui prend son essor dans la dérivation de produits.*
La gestion de l'évolution de la *ligne de produits* est traitée sur la base de l'infrastructure introduite par le langage PLiMoS. La proposition d'évolution incrémentale par extension repose sur un processus outillé centré sur la phase de dérivation des produits.

1.4 Organisation du mémoire

Le mémoire prend ses racines dans l'état de l'art du domaine, qui permet de répondre à plusieurs questions préliminaires de recherches : (QR1) Quel est le cadre théorique et quels sont les concepts clés ?, (QR2) Quels sont les artefacts manipulés ?, (QR3) Quelles sont les particularités d'une évolution d'une *ligne de produits* ? Ces questions révèlent *3 recommandations* et *4 besoins*. Ces fondations permettent ensuite d'exposer les contributions 1 à 5 dans des parties dédiées avant de conclure. La figure 1.1 illustre la structuration du tapuscrit en y faisant apparaître les questions et contributions. Dans la suite du mémoire, un certain nombre de citations sont directement extraites des sources, la plupart sont en anglais et une traduction de ces citations en français est renseignée en annexe A.

L'exposé de cette thèse est structurée en cinq parties majeures, accompagnées de parties structurant le document (bibliographie, sitographie, annexes, indexes et glossaire). Les contributions de la thèse sont principalement portées par les parties 2 et 3, et validées dans la 4.

1.4.1 Première partie : Modélisation et évolution des Lignes de Produits

La première partie, intitulée "Modélisation et évolution des Lignes de Produits" est consacrée à l'état de l'art et à l'élicitation de la motivation, et comporte quatre chapitres distincts :

- Le chapitre 2 présente le cadre théorique de l'Ingénierie Dirigée par les Modèles et ses concepts clés : l'abstraction et la séparation des préoccupations. Ce chapitre insiste sur l'aspect relationnel de la modélisation et présente la notion de modélisation intentionnelle.
- Le chapitre 3 introduit la modélisation de la variabilité afin de présenter une partie des artefacts manipulés dans cette thèse. Une étude poussée des relations entre des modèles suivant différentes préoccupations est menée dans ce cadre de la modélisation de la variabilité.
- Le chapitre 4 décrit les *lignes de produits* et ses concepts fondamentaux. La division de deux ingénieries est présentée - du Domaine et de l'Application - afin de présenter, d'une part, l'architecture des Lignes de Produits et, d'autre part, la dérivation de produits. Enfin, ce chapitre fait un état de l'art de l'évolution des LdPs. Il se focalise sur les approches proactives et réactives d'évolution des *lignes de produits* en introduisant les processus itératifs.
- Le chapitre 5 réalise la synthèse des besoins relevés dans l'état de l'art, détaille le problème, et énonce les motivations et contributions de cette thèse.

1.4.2 Deuxième partie : PLiMoS, une modélisation de l'espace de modélisation des *lignes de produits*

La deuxième partie, intitulée "PLiMoS, une modélisation de l'espace de modélisation des *lignes de produits*", comporte quatre chapitres et présente les contributions de la thèse. Ces chapitres

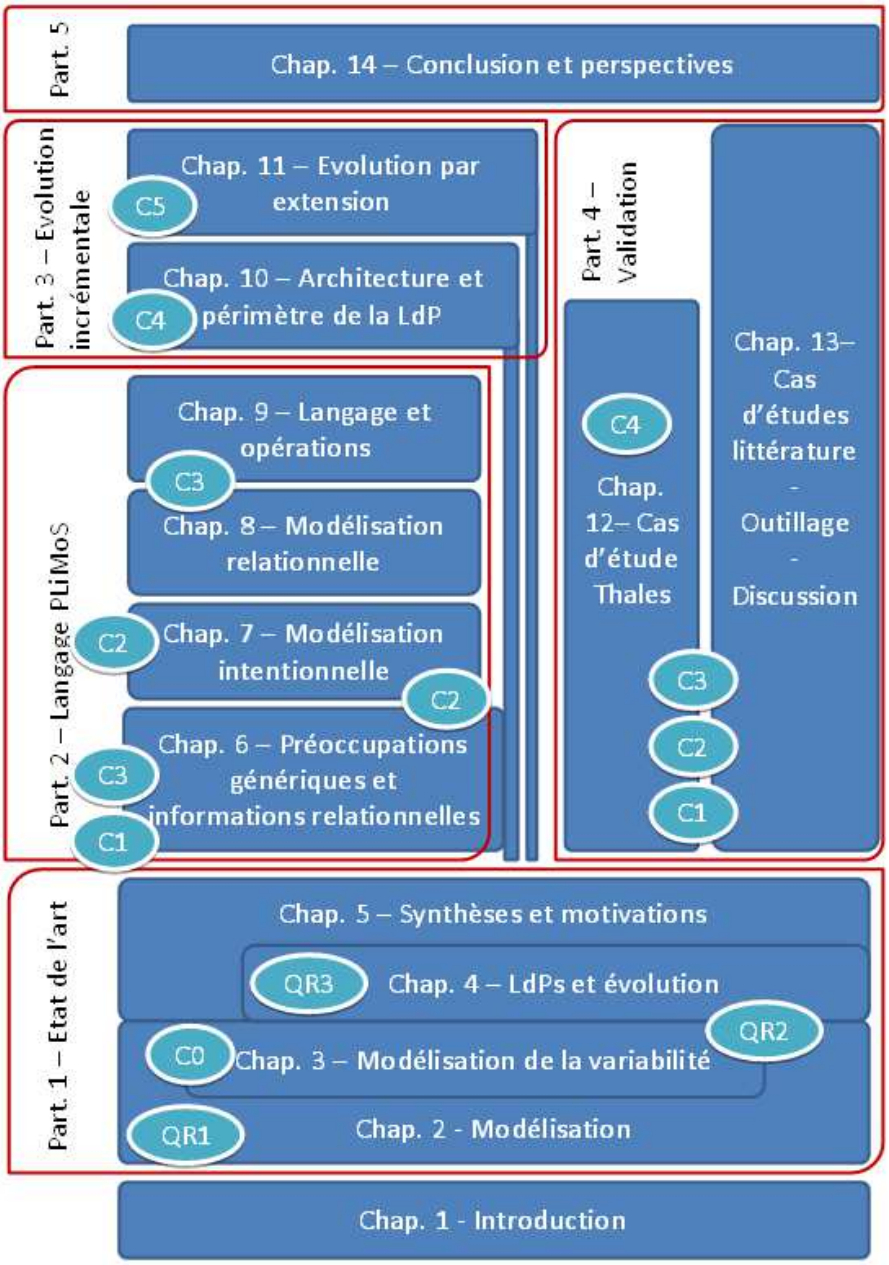


Figure 1.1: Plan du mémoire de thèse

sont des contributions à l'ingénierie du domaine et plus particulièrement à la modélisation de la variabilité, de l'agencement des modèles et de leurs connexions.

- Le chapitre 6 fournit une description du cadre conceptuel générique de décomposition multidimensionnelle de la variabilité répondant à une problématique d'ingénierie système. La décomposition multi point de vue et le cadre conceptuel à 4 dimensions y sont introduits. De plus, une réponse au manque d'expression claire et précise de l'information relationnelle est proposée, au travers l'introduction d'un langage de modélisation dédié appelé PLiMoS.
- Le chapitre 7 présente une modélisation intentionnelle de l'espace de modélisation de la *ligne de produits* et du langage de modélisation la soutenant (sous ensemble intention de PLiMoS). Le chapitre explicite les rôles, interdépendances et usages des artefacts de l'espace de modélisation.
- Le chapitre 8 expose les différentes facettes de la modélisation des relations avec le langage PLiMoS, qui type et réifie les dépendances inter et intra modèles de l'espace de modélisation. Le langage PLiMoS se tisse sur des langages de variabilité existants pour clarifier et renforcer la sémantique des relations entre artefacts modélisés.
- Le chapitre 9 énonce les opérateurs du langage PLiMoS et détaille les activités liées à la modélisation de la variabilité avec celui-ci.

1.4.3 Troisième partie : Évolution incrémentale de la modélisation des *lignes de produits*

La troisième partie comporte deux chapitres et aborde les contributions de la thèse relatives à l'évolution de la *ligne de produits*. Les chapitres sont des contributions à l'ingénierie du domaine (chapitre 10) et de l'application (chapitre 11).

- Le chapitre 10 introduit les considérations architecturales et l'application d'un patron d'architecture dédié à la Ligne de Produits, formant ainsi une plate-forme conceptuelle impactant la modélisation de la variabilité.
- Le chapitre 11 présente le processus d'évolution par extension d'une *ligne de produits*. Les évolutions possibles sont caractérisées et le processus, en lien avec la dérivation de produits, est exposé. L'approche vise à rapprocher les ingénieries de l'application et du domaine pour en extraire des bénéfices en termes de flexibilité et de factorisation des coûts.

1.4.4 Quatrième partie : Application et applicabilité de la modélisation

La quatrième partie comporte quatre chapitres qui concernent (i) la validation des contributions de la partie II sur la modélisation de l'espace de modélisation, (ii) la validation des contributions de la partie III sur l'architecture et l'évolution incrémentale par extension de la Ligne de Produits, (iii) un bilan des apports et limites des approches.

- Le chapitre 12 présente une étude de cas industriel dans l'ingénierie du logiciel et montre l'adéquation et la cohérence de la méthodologie proposée avec le processus de développement en place à Thales. Ce chapitre concerne la modélisation de l'espace de modélisation et l'architecture.
- Le chapitre 13 présente une série de cas d'études issus de la littérature et présente leur modélisation avec PLiMoS. Il introduit également l'outillage réalisé concernant la modélisation de l'espace de modélisation, et l'évolution. Enfin, il réalise une synthèse et une discussion concernant la validation.

1.4.5 Cinquième partie : Conclusion et perspectives

Enfin, dans une dernière partie, le chapitre 14 conclut en établissant un bilan des contributions apportées par cette thèse. Des perspectives offertes par ces travaux sont détaillées et des voies de recherche proposées.

Première partie

Modélisation et évolution des *lignes de produits*

"Τὸ δὴ μετὰ τοῦτο συνεπισπώμεθα τῷ λόγῳ τῇδε σκοποῦντες, μὴ περὶ πάντων τῶν εἰδῶν, ἵνα μὴ ταπαττώμεθα ἐν πολλοῖς, ἀλλὰ προελόμενοι τῶν μεγίστων λεγομένων ἄττα, πρῶτον μὲν ποῖα ἕκαστά ἐστιν, ἔπειτα κοινωνίας ἀλλήλων πῶς ἔχει δυνάμεως, ἵνα τό τε ὄν καὶ μὴ ὄν εἰ μὴ πάσῃ σαφηνείᾳ δυνάμεθα λαβεῖν, ἀλλ΄ οὖν λόγου γε ἑνδεεῖς μηδὲν γιγνώμεθα περὶ αὐτῶν, καθ΄ ὅλον ὸ τπόπος ἐνδέχεται τῆς νῦν σκέψεως," [a]

[a] Poursuivons maintenant en examinant non pas toutes les idées, de peur de nous perdre dans cette multitude, mais quelques-unes choisies parmi celles que l'on appelle les plus grandes ; considérons d'abord ce quelles sont chacune à part, ensuite jusqu'à quel point elles sont susceptibles d'être associées les unes avec les autres, afin que si nous ne pouvons pas obtenir une connaissance parfaitement claire de l'être et du non-être, du moins nous ne nous trouvions pas dans l'impossibilité d'en rendre compte, dans les limites mêmes de la question que nous nous sommes posée - trad. V. Cousin

— Platon, *Le Sophiste*

CHAPITRE 2

Modélisation et Ingénierie Dirigée par les Modèles

"We should reconcile ourselves with the fact that we are confronted, not with one concept, but with several different concepts which are denoted by one word ; we should try to make these concepts as clear as possible to avoid further confusions, we should agree to use different terms for different concepts ; and then we may proceed to a quiet and systematic study of all the concepts involved."

— Alfred Tarski, *The Semantic Conception of Truth and the Foundations of Semantics, 1944*

Pour relever le défi de la complexité croissante, la communauté du génie logiciel voit émerger, voilà quelques années, l'*ingénierie dirigée par les modèles* ([Ken02, CESW04, FR07]). Le terme "modèle" présente une importante variété d'usages dans les sciences. À travers cette diversité des usages et des domaines, le sens oscille entre concret et abstrait, figuration et formalisme, image et équation, échantillon et étalon, réalisation matérielle et norme abstraite. Paradoxalement, cette équivocité essentielle est le trait permanent à travers le large spectre des différents usages. L'IDM place le modèle au centre des débats et fait la part belle à l'abstraction et à la séparation des préoccupations pour tenter de canaliser, résorber et maîtriser la complexité intrinsèque des systèmes à développer. L'IDM marque l'émergence au plan de la réflexion d'une prise de conscience bien plus ancienne de l'importance de la formalisation du langage et de l'abstraction dans le génie logiciel. Basiquement, dans l'approche est développée une suite de modèles en suivant une séparation des spécifications fonctionnelles des détails techniques ou spécifiques à une plate-forme, ce qui se rapproche du raffinement itératif proposé par Wirth quelques années plus tôt [Wir71].

Ce chapitre répond à la question de recherche suivante :

QR 1. *Quel est le cadre théorique et quels sont les concepts clés ?*

L'abstraction est un concept qui concentre l'intégralité du fondement des questions portant sur la nature de la pensée (section 2.1.1). L'IDM inscrit l'artefact de modélisation comme base conceptuelle à l'aide de laquelle toute activité de raisonnement se déploie (section 2.1). Les modèles sont une représentation simplifiée d'un aspect de la réalité pour un objectif donné et se retrouvent impliqués dans des méthodologies et des processus de développement (section 2.2). La formation et la formalisation des ensembles cognitifs de définition de ces artefacts de modélisation se retrouvent sous la problématique de métamodélisation. L'ensemble des artefacts (objets, modèles, langages, définitions) constituent l'espace de modélisation [Fav05b] (section 2.3). Les raisonnements et analyses sont réalisés dans des espaces technologiques dédiés [DGD06], offrant un cadre méthodologique et outillé adapté pour réaliser des opérations sur les modèles (transformations, générations, compositions, comparaisons, validations, etc.) (section 2.4). À l'instar des développements "classiques", l'IDM doit faire face à des réalités de maintenance et d'évolution et peut se lier à des approches de traçabilité (section 2.5).

2.1 Des modèles de l'IDM

L'abstraction et la séparation des préoccupations constituent les apports fondamentaux de l'IDM. Elles permettent au concepteur de se focaliser sur l'objectif à résoudre, avec un minimum de perturbations provenant de considérations autres que les siennes. Par delà, l'objectif de cette approche est de résoudre les problèmes d'intégration et d'interopérabilité[1]. Avec de telles approches,un changement d'intergiciel n'impacte pas le modèle - stable - de l'application à un niveau d'abstraction donné mais uniquement les générateurs de code.

Cette section définit la notion d'abstraction (section 2.1.1) pour faire ressortir dans un second temps les propriétés intrinsèques des modèles (section 2.1.2). En effet, le mot est doté d'une grande polysémie qui lui permet d'être employé dans des contextes très différents, et qui se doit d'être précisé dans le cadre de l'IDM. Enfin, la notion de modularité fournie par la *séparation des préoccupations* et par les *points de vues* est présentée (section 2.2).

2.1.1 De l'abstraction

L'abstraction est l'une des clefs pour maîtriser la complexité des logiciels. Le mot abstraire est erroné s'il est pris comme un détachement total des choses, et il n'exprime que la moitié de ce qu'il signifie s'il est compris comme isolé de l'ensemble de la représentation. L'article de l'*Encyclopædia Universalis* [Del03] propose quatre nuances au terme abstraction, qui sont :

1. L'*abstraction par simplification*, une remontée au principe de la mise à l'écart d'une réalité trop complexe pour être pensée de façon unitaire dans sa diversité. Ainsi entendue, l'abstraction est une opération préalable et nécessaire à toute idéation. Il s'agit de ne pas s'arrêter aux particularités d'un être et en abstraire inductivement une loi.
2. L'*abstraction par généralisation*, qui est une opération mentale nécessaire à la dynamique de la conceptualisation. Elle se fonde sur la structure même de la logique des classes et les propriétés opératoires du langage humain. Il s'agit de remonter des éléments vers des propriétés plus générales.
3. L'*abstraction par sélection*, une opération nécessaire et propédeutique à toute classification. Cette forme d'abstraction entraîne un appauvrissement du réel. L'abstraction par sélection est de vocation réductrice. Elle tend à réduire toute particularité à un trait différentiel, toute singularité à un jeu de déterminismes circonstanciels, tout individu à un élément dans un ordre, toute totalité à une somme de parties. L'abstraction désigne ici le processus mental

[1] "It's about integration. It's about interoperability" [OS00, Préface]

qui consiste, en partant d'une donnée quelconque, à isoler un trait spécifique quelconque, une détermination considérée en elle-même. La spécificité de cette forme d'abstraction réside dans le fait d'isoler de toute intégralité ou globalité de l'objet un aspect fragmentaire local.

4. L'*abstraction de schématisation*, qui permet une modélisation d'une donnée et sa formalisation. Elle permet d'extraire du sensible des éléments de forme (de l'information), à partir desquels, par des abstractions successives de même type, se construiraient des étages de complexité, formés par des représentations et des ensembles cognitifs de plus en plus symboliques.

Alors que la première signification trouve son utilité avec les mathématiques et l'abstraction qui est réalisée des environnements physiques et l'établissement d'analogies (processus d'idéation), les trois autres trouvent une réalité dans le domaine de l'informatique. L'*abstraction par généralisation* correspond au processus de cognition de conceptualisation et trouve une application dans la Programmation Orientée Objet (POO) par exemple ; l'*abstraction par sélection* correspond à une classification, avec par ex. une application dans les ontologies (il est à noter que la généralisation est transitive alors que la classification ne l'est pas) ; enfin l'*abstraction de schématisation* correspond au processus même de modélisation. Toutes trois sont partie prenantes du processus de modélisation informatique et de métamodélisation de l'IDM.

Un langage informatique est une abstraction en lui-même, puisqu'il fait correspondre à un langage ésotérique (le langage machine) un autre langage mieux adapté à la compréhension humaine. En outre, est nommée également aussi couche d'abstraction matérielle une couche logicielle accédant au matériel informatique. Plus généralement, est nommée couche d'abstraction toute couche logicielle cachant au développeur l'implémentation de la couche inférieure, lui évitant de fournir une implémentation différente selon les spécificités de la couche inférieure.

De nombreux courants de l'informatique utilisent un pan de l'abstraction permettant la modélisation d'une réalité, à l'exemple de Giunchiglia et Walsh [GW92] dans le domaine de l'intelligence artificielle qui définissent l'abstraction telle :

> *"[...] abstraction is the mapping from one representation of a problem to another which preserves certain desirable properties and which reduces complexity."*
>
> Giunchiglia et Walsh [GW92] (traduction p. 203)

Cette définition, comme d'autres définitions pragmatiques dans le domaine informatique, place l'abstraction dans un contexte de modélisation en y mixant la généralisation et la sélection et en y corrélant la propriété de compréhensibilité.

2.1.2 Des modèles

L'IDM se caractérise par la relation au monde (relation objet modèle) selon un mode d'appréhension par concept ; l'approche n'étudie pas les constructions : c'est-à-dire "les idées elles-mêmes" mais les "objets au-delà des idées". Un modèle n'est pas le monde réel, mais un artefact artificiel permettant une meilleure compréhension de systèmes réels.

La notion de modèle a reçu un emploi très large dans la méthodologie des sciences et l'Ingénierie Dirigée par les Modèles place les modèles au centre des développements des systèmes. À ce titre, il est légitime de s'interroger sur la définition de ce M de l'IDM.

Définitions

Paradoxalement, il n'existe pas de consensus fort de la communauté autour de l'adhésion à une définition unique de modèle, sans pour autant marquer d'énormes divergences. Pour preuves,

le tableau 2.1 présente des définitions issues de la littérature IDM. Ces définitions sont, pour certaines, limitantes et font face aux critiques. Concernant la définition de [OMG05], Favre dénonce dans [Fav05b] le fait que la modélisation n'est pas l'apanage de système physique. En outre, les définitions qui considèrent l'abstraction comme identique à une "simplification" sont trop restrictives car introduisant un biais sémantique.

Tableau 2.1: Définitions de modèle dans l'IDM.

[MoTAIL65]	*To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A.*
[BRJ99]	*A model is a simplification of reality [...] so that we can better understand the system we are developing.*
[BG01]	*A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system.*
[Jac02]	*Here the word Model means a part of the Machine's local storage or database that it keeps in a more or less synchronised correspondence with part of the Problem Domain. The Model can then act as a surrogate for the Problem Domain, providing information to the Machine that can not be conveniently obtained from the Problem Domain itself when it is needed.*
[Lud03]	*Models help in developing artefacts by providing information about the consequences of building those artefacts before they are actually made.*
[OMG03b]	*A model of a system is a description or specification of that system and its environment for some certain purpose.*
[Sei03]	*A model is a set of statements about some system under study (SUS).*
[Sel03]	*Engineering models aim to reduce risk by helping us better understand both a complex problem and its potential solutions before undertaking the expense and effort of a full implementation.*
[Bro04]	*Models provide abstractions of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on the relevant ones.*
[Küh06]	*A model is an abstraction of a (real or language based) system allowing predictions or inferences to be made.*

traductions p. 203

Les modèles, en faisant abstraction des détails du réel qui sont nuisibles à l'intellection, favorisent la compréhension des concepts et de leurs associations et donc la compréhension de la réalité. Ces modèles, qui ont une substituabilité bornée, sont usuellement analytiques et la pensée du concepteur est, conséquemment, à distance de ce sur quoi elle porte. Le principe de substituabilité s'exprime comme suit : le modèle doit être suffisant et nécessaire pour permettre de répondre à certaines questions en lieu et place du système qu'il représente [MoTAIL65].

Dans les hypothèses de cette thèse, la définition suivante est prise pour référence :

Définition 1. *Modèle : Un modèle est une (un ensemble d') abstraction(s) sémantiquement close(s) d'un système, construit dans une intention particulière. Dans ce cadre le modèle doit être substituable au système modélisé.*

Le modèle réalise une abstraction schématisante et éventuellement des abstractions de sélections et généralisation ; un modèle peut encapsuler plusieurs niveaux d'abstraction.

Types et propriétés des modèles

Seidewitz [Sei03] propose une classification duale des modèles, à savoir : descriptif et prescriptif. Sur le sujet, Muller *et al.* évoquent respectivement une nature de modèle analytique et synthétique [MFBC10]. Les significations sont les suivantes :

- *Modèle descriptif* : Le modèle réfère à une réalité existante. Tous les modèles explicatifs liés à l'observation sont descriptifs [Mon08].
- *Modèle prescriptif* : Le modèle considère un système qui n'existe pas encore. En ce sens il peut être qualifié de spécification.

De surcroît, il est réalisé une distinction entre des modèles :

- *statiques*, ou dit structurels, par ex. dans UML 2 [OMG11d, OMG11c] : de classes, d'objets, de composants, de déploiement, de paquetages, de structure composite, et de profils ;
- *dynamiques*, qui sont soit (i) comportementaux, par ex. dans UML 2 [OMG11d, OMG11c] : de cas d'utilisation, d'états-transitions, d'activité ; soit (ii) d'interaction : de séquence, de communication, de vue d'ensemble des interactions, et de temps.

Les modèles peuvent être exécutables s'il leur est associé une sémantique d'exécution (par ex. par un langage d'action dans le cas de modèles statiques).

En outre, selon Selic [Sel03], un modèle d'ingénierie doit posséder les cinq qualités suivantes :

- *abstraction* : Un modèle doit enlever ou masquer des détails, il s'agit dans cette description d'une abstraction par sélection ;
- *compréhensibilité* : Un modèle doit être intuitif et la compréhension du système par le modèle doit être facilitée par ce dernier ;
- *exactitude* : Un modèle doit fournir une représentation exacte et juste des concepts d'intérêt du système ;
- *prédictibilité* : Un modèle doit posséder la faculté de prédire les propriétés réelles du système ;
- *modicité (du coût)* : Un modèle doit être significativement moins cher à construire et à analyser que le système lui même. Un modèle permet donc de raisonner sur un système à moindre coût et, dans certains cas, permet même d'étudier des systèmes impossibles à construire ou à tester.

La logique est concernée en ce que les modèles donnent matière à l'étude sémantique, qui s'intéresse aux rapports des signes avec la réalité, et à l'étude pragmatique, qui reconnaît les motifs du choix des langages obéissant à des raisons d'efficacité, d'optimalité (cf. section 2.2). Ces modèles doivent être conformes à un langage manipulable informatiquement et se retrouvent liés à d'autres modèles (cf. section 2.3).

2.2 Séparation des préoccupations

Le concept de séparation des idées ou séparation des préoccupations *(Separation of Concerns - SoC)* [HL95] a très tôt été considéré comme un artefact essentiel pour maîtriser la complexité essentielle des développements logiciels. Il s'agit pragmatiquement d'une application de la stratégie générale du "diviser pour régner". Les idées sous-jacentes des SoC proviennent de E. W. Dijkstra :

> *"We know that a program must be correct and we can study it from that viewpoint only ; we also know that it should be efficient and we can study its efficiency on another day, so to speak. In another mood we may ask ourselves whether, and if so : why, the program is desirable. But nothing is gained -on the contrary !- by tackling these various aspects simultaneously. It is what I sometimes have called "*the separation of concerns*", which, even if not perfectly possible, is yet the only available technique for*

> *effective ordering of one's thoughts, that I know of. This is what I mean by "focusing one's attention upon some aspect" : it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant. It is being one- and multiple-track minded simultaneously."*
>
> E. W. Dijkstra [Dij82] (traduction p. 204)

La séparation des préoccupations apparaît dans les différentes étapes du cycle de vie du logiciel et prend donc une multitude de formes. Il peut s'agir de la séparation dans le temps qui concerne le traitement de la conception à la réalisation des différentes facettes du logiciel qui sont alors abordées successivement lors du processus de développement.

L'IDM essaie de lire le complexe réel sous l'apparence "simple" fournie par les modèles à différents niveaux d'abstraction dans les développements. Les concepteurs se focalisent sur des artefacts dans un spectre réduit de préoccupations à l'aide de (i) langages génériques (par exemple UML [OMG11d, OMG11c]), ou spécifiques à leur domaine (*Domain Specifique Languages - DSLs*, ou parfois appelés *Domain Specifique Modeling Languages - DSMLs*), cf. section 2.2.1, et (ii) de points de vue (encapsulation d'informations ciblées sur le métier de l'utilisateur), cf. section 2.2.2. La légitimité des points de vue tient, évidemment, à leur intelligibilité et à leur communicabilité.

Il faut en voir l'utilité dans un contexte très général : un modèle à un niveau d'abstraction donné peut être abordé de plusieurs points de vue transverses et spécifiques à un domaine métier, selon que l'observateur considère, par exemple, ses propriétés de sûreté de fonctionnement, de performance, de persistance, de sécurité ou de tolérance aux pannes.

La séparation des préoccupations propose un support méthodologique lors de la conception d'une application. L'identification des facettes tend à une séparation orthogonale des préoccupations pour améliorer la cohérence et le couplage. La séparation des préoccupations est, à l'instar de la modularité et l'abstraction, un des principes fondamentaux du génie logiciel [GJM03].

2.2.1 Méthodologies et niveaux d'abstraction

Le premier principe utilisé pour maîtriser la complexité de la conception de logiciels consiste à augmenter l'abstraction avec laquelle on aborde un problème (cf. section 2.1.1). Tel qu'énoncé précédemment, le principe d'abstraction ne considère ainsi que les informations d'un système jugées importantes à un moment donné, en omettant les autres informations. Bien que très subjective, une "bonne" abstraction se caractérise par le fait de n'adresser qu'une facette de la construction d'une application. Cette facette doit être suffisamment orthogonale aux autres afin de pouvoir être isolée et composée.

La mise en œuvre des niveaux d'abstraction peut être effective par des langages dédiés (*DSLs*) [vDKV00, CM98] ou également par des méthodologies IDM prônant l'utilisation de plusieurs niveaux d'abstraction, telle que la méthodologie MDA [OS00] ou ARCADIA [Voi08] qui reposent sur des langages génériques (par exemple UML), ou spécifiques (*DSMLs*).

Le principe d'abstraction propose de maîtriser la complexité d'une application en fournissant une représentation de haut niveau de l'application davantage pérenne et compréhensible par l'ensemble des acteurs d'un projet informatique, en l'épargnant d'une obsolescence des plates-formes technologiques.

Le MDA

L'approche *Model-Driven Architecture - MDA* a été introduite en 2000 par l'OMG [OS00, OMG01] avec pour objectif de répondre au susnommé problème de prolifération des intergiciels (*"middleware proliferation problem"*) basé sur des technologies distinctes, par ex. .NET [Ric02], J2EE [16],

CORBA [OMG06]. L'approche tend à promulguer de bonnes pratiques de modélisation et à exploiter pleinement les avantages des modèles. Elle propose de modéliser la plate-forme technologique (*Platform Description Model - PDM*) et de la séparer des autres décisions de conception, permettant dès lors de nombreuses projections d'un modèle système réutilisable sur des technologies cibles. La démarche s'organise selon un cycle de développement "en Y" comme l'illustre la figure 2.1. Selon la terminologie MDA, une plate-forme est définie telle :

> *"[...] a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented."*
>
> MDA Guide [OMG03b] (traduction p. 204)

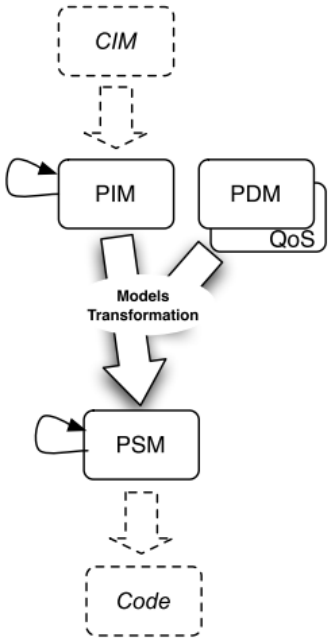


FIGURE 2.1: Méthodologie MDA [OS00, OMG01].

Une séparation des préoccupations en trois niveaux de modélisation d'architecture y apparaît : (i) CIM (*Computation Independent Model*), qui définit les concepts métiers et se centre sur les exigences, (ii) PIM (*Platform Independent Model*), qui modélise les résultats de l'analyse et de la conception, et (iii) PSM (*Platform Specific Model*), un modèle de niveau code d'implémentation. La démarche MDA s'organise selon un cycle de développement en Y ; le MDA a fait l'objet d'un grand intérêt dans la littérature spécialisée, par ex. [KWB03]. Cette approche vise à mettre en valeur les qualités intrinsèques des modèles, telles que pérennité, productivité et prise en compte des plate-formes d'exécution. Le MDA inclut pour cela la définition de plusieurs standards, notamment UML [OMG11d, OMG11c], MOF [OMG11a] et XMI [OMG11b].

La méthodologie ARCADIA

Le cadre de développement des applications à Thales est orienté modélisation et suit la méthodologie présentée dans [Voi08]. Cette méthodologie, nommée ARCADIA (pour *ARChitecture Analysis & Design Integrated Approach*) encadre les développements en proposant des activités de modélisation qui sous-tendent les développements système et logiciel pour les exigences, la conception conceptuelle et détaillée et l'implémentation.

La méthodologie identifie deux dimensions distinctes, la première relative aux phases du cycle de développement et donc dans une approche de modélisation aux divers niveaux d'abstraction. La seconde concerne une représentation en vues métiers (propriété non fonctionnelle, par ex. performance, propagation de fautes). En effet, l'approche ARCADIA met également en avant l'utilisation d'une description de l'architecture par points de vues (basée sur les fondations conceptuelle de l'ISO/IEC 42010 [ISO11]).

Le contexte industriel de modélisation présente les niveaux d'abstraction suivants :

Contexte : les préoccupations à ce niveau sont duales, recouvrant une analyse des besoins des utilisateurs (besoins et objectifs, missions et activités attendues) ainsi qu'une analyse des besoins du système. La première a pour vocation d'assurer une bonne adéquation entre la définition du système au regard de son utilisation opérationnelle réelle et définit les conditions d'IVVQ (intégration, validation du besoin client, vérification en regard des exigences du système et qualification) ; les artefacts résultants sont des "Use-Cases" et des scenarii. La seconde analyse se concentre sur la manière dont les besoins opérationnels déterminés à l'étape précédente peuvent être satisfaits par le système, avec ses comportements et ses qualités requises ; les artefacts résultants sont des besoins fonctionnels et des contraintes non-fonctionnelles.

Logique : à ce niveau sont concernées les analyses logiques et la réalisation de modèles indépendants de la plate-forme, une identification des modules système, en excluant leur implémentation et les questions techniques / technologiques.

Physique : à ce niveau sont adressées les considérations de plate-forme et leur intégration dans la conception de l'architecture. Le modèle de sortie est une architecture physique de composants à produire.

La figure 2.2 illustre cette approche ; les détails sont disponibles dans [Voi08]. La figure représente également un dernier niveau de modélisation non listé ci-dessus, car hors problématique de cette thèse, relatif à la définition des contrats de réalisation des EPBS (*End-Product Breakdown Structures*) pour une construction hiérarchique.

2.2.2 SoC, paradigmes et points de vue

Un autre type de séparation concerne la séparation des caractéristiques. En, effet, sur les fondements établit par Dijkstra, de nombreuses approches et paradigmes de programmation ont émergé, considérant les programmes comme une composition de préoccupations élémentaires ou caractéristiques. Pour en citer quelques uns : Tarr *et al.* proposent une décomposition en de multiples dimensions de préoccupations [TOHS99] pour aller vers une programmation en "hyperslices" ; la programmation orientée aspect (*Aspect-oriented programming - AOP*) [KLM+97], qui se focalise sur le traitement des préoccupations transversales ; et plus proche des LdPs, la programmation orientée *features* (*Feature Oriented Programming - FOP*) [Pre97]. Enfin, Jézéquel [Jéz08] explicite le lien entre l'IDM et les approches fondées sur les aspects et avance l'argument que la modélisation et le tissage d'aspects sont des activités symétriques.

Une troisième illustration du principe de séparation des préoccupations est la séparation des "vues" d'un système. Il peut s'agir, par exemple, d'un point de vue :

1. fonctionnel décrivant le comportement fonctionnel et nominal du système ;

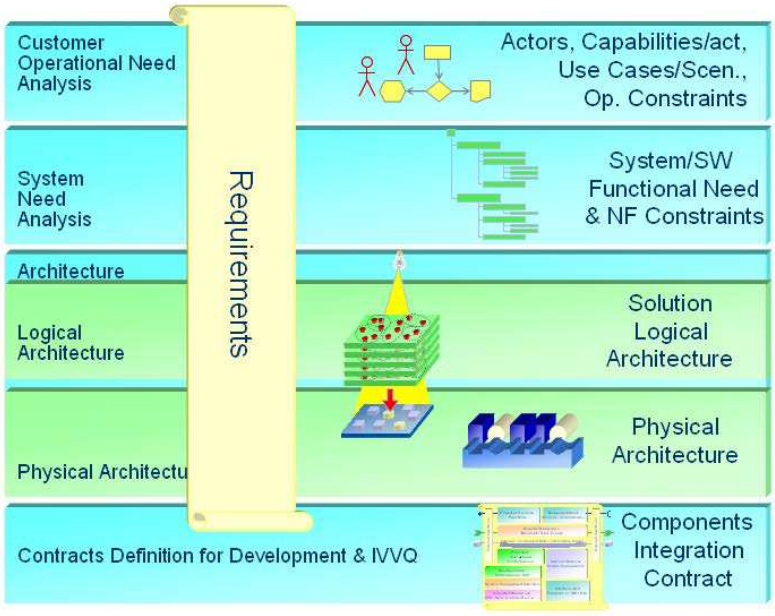


FIGURE 2.2: Méthodologie ARCADIA [Voi08].

2. de tolérance aux défaillances explicitant les comportements en cas de pannes ;
3. d'évaluation de performances permettant de calculer les latences, charges, débits et autres caractéristiques temps réel, de modèles de robustesse aux perturbations mécaniques, électromagnétiques, etc.

Les points de vue sont spécialisés, définis avec une sémantique propre au domaine métier. Par exemple, dans le contexte de l'ingénierie des systèmes avioniques, l'article [BFW01] décrit un système dans quatre domaines métiers (c.-à-d. quatre points de vue) : fonctionnel, Sûreté de Fonctionnement (SdF), performances temps réel et résistance électromagnétique.

Concernant l'architecture d'un système logiciel, apparaissent très clairement plusieurs utilisateurs et parties-prenantes, intéressés par différentes facettes du système et de son éventuel déploiement/usage. Plusieurs vues d'architecture du système sont définies, par ex. [IEE00]. Une approche populaire de multi-vues architecturales vient de la méthodologie des "4+1" vues [Kru95] proposée par Kruchten pour une modélisation avec UML (logique, processus, physique, développement, et les scenarii de cas d'utilisation).

La gestion des points de vue amène irrémédiablement une problématique de gestion de cohérence entre ces vues, source de nombreuses recherches, par ex. [FGH+94, NEFE03].

La construction d'un point de vue à partir du modèle global peut se faire par une sémantique de traduction (transformations de modèles [JGB06]) vers un espace technologique formellement défini [CESW04] (par exemple Altarica pour la sûreté de fonctionnement, le formalisme des files d'attente pour l'évaluation de performances, etc.), chaque langage spécifique embarquant quant à lui et à son niveau sa syntaxe et sa sémantique propres. Cette approche permet de bénéficier d'un environnement spécifique au point de vue, comprenant de l'outillage. Néanmoins, se pose le problème récurrent de l'interprétation de l'exécution et de l'analyse du modèle dans le langage source : quid de la remontée des résultats obtenus vers l'espace d'origine ? Les approches de SdF ou de validation (*model-checking*), nécessitent une translation sémantique et un retour dans l'espace d'origine, construit dans un premier temps de manière *ad hoc*.

2.3 De l'espace de modélisation

Les sections précédentes ont introduit les modèles et les différents niveaux d'abstractions auxquels ils se placent. Qu'en est-il de l'environnement du modèle, de son moyen de le définir et de représenter ses interactions avec d'autres artefacts ? Cet ensemble d'artefacts constitue l'espace de modélisation [DGD06].

Cette section introduit la métamodélisation et les concepts associés, les espaces techniques et les relations entre les modèles, pour enfin présenter une approche de relation intentionnelle.

2.3.1 Métamodélisation

La définition de métamodèle n'est pas triviale et la question fait l'objet de discussions, entre autres [Sei03, Fav05b, Fav05a, Fav06, Küh06].

Kuehne explore la nature du préfixe méta (μετά) qui exprime la réflexion, c.-à-d. l'application d'un concept à soi-même. En suivant ce raisonnement, un métamodèle est un modèle de modèles. Cependant Kuehne et Favre montrent l'existence de plusieurs relations de *modèle-de* (ontologique ou linguistique [Küh06]), et selon Favre la réponse à la question "le métamodèle est-il un modèle d'un modèle" n'est pas unique et ne peut être tranchée aisément :

> *"Is a metamodel a model of a model ?* [...] *the answer is in the middle, or better said, one can say either YES or NO, depending on the interpretation of the word model."*
>
> J.-M. Favre [Fav06] (traduction p. 204)

Dans le cadre de cette thèse, la définition de métamodèle s'accorde sur celle de l'OMG [OMG11a] et est explicitée comme suit :

Définition 2. *Métamodèle : Un métamodèle est un modèle qui définit le langage d'expression d'un modèle, c.-à-d. le langage de modélisation.*

Le métamodèle (MM_L) consiste en l'association d'une syntaxe abstraite AS_L au domaine sémantique $\mathcal{S}_L$ par une application $\mathcal{M}_L$, telle que $\mathcal{M}_L : AS_L \rightarrow \mathcal{S}_L$, soit :

$$MM_L \triangleq \langle AS_L, \mathcal{S}_L, \mathcal{M}_L \rangle$$

Le métamodèle représente les concepts du langage utilisés et la sémantique qui leur est associée. La conformité du modèle au métamodèle est une relation de premier ordre dans l'IDM, elle est le pendant de la relation d'*instance-de* du paradigme objet.

Dans un contexte de modélisation, le métamodèle constitue la base des langages de modélisation *(Domain Specific Modeling Languages)*, qui consiste en une notation syntactique (syntaxe) associée à une signification (sémantique). Les syntaxes concrètes (CS_L) sont textuelles ou diagrammatiques, plusieurs fonctions d'association M_L peuvent être liées à une syntaxe abstraite donnée. La fonction d'association est définie comme : $M_L : AS_L \rightarrow CS_L$; et le langage de modélisation s'exprime dès lors sous la forme :

$$\mathcal{L} \triangleq \langle AS_\mathcal{L}, \mathcal{S}_\mathcal{L}, \mathcal{M}_\mathcal{L}, CS_\mathcal{L}, M_\mathcal{L} \rangle$$

2.3.2 Espaces techniques et méta-métamodèles

Le langage de métamodélisation doit également être défini. Les travaux présentés par la suite se placent dans une approche de modélisation stricte, de hiérarchie de niveaux linéaire, où tous les éléments définis à un niveau d'abstraction sont évalués au regard du niveau strictement supérieur (*strict metamodeling*, sémantique de relation d'instanciation de type *shallow instantiation*).

> *"**Strict Metamodeling** : In an $n-level$ modeling architecture, M_0, M_1, ..., M_{n-1}, every element of an M_m level model must be an instance-of exactly one element of an M_{m-1} level model, for all $0 \leq m < n-1$, and any relationship other than the instance-of relationship between two elements X and Y implies that level(X)= level(Y)."*
>
> Atkinson et Kühne [AK02] (traduction p. 204)

L'approche inverse (*deep instantiation*, *relaxed strictness*) suit le courant initié par Atkinson et Kühne sur la modélisation multi-niveaux, poursuivi dans des approches liant les modèles de processus et d'artefacts [AK01, dLG10, HS11].

La réponse pragmatique de l'OMG à la question de langage de définition du métamodèle est de proposer un langage sous la forme d'un modèle : le méta-métamodèle MOF (Meta-Object Facility) [OMG11a]. De part sa nature de modèle, ce dernier doit également être défini à partir d'un langage de modélisation. Afin de limiter le nombre de niveaux d'abstraction, le méta-métamodèle est méta-circulaire, c.-à-d. qu'il possède la capacité de se décrire lui-même. Bien que décrié par certains (par ex. Seidewitz [Sei03] ou Geisler *et al.*) [GKP98]), ces derniers prônant une approche à niveau supérieur axiomatique (spécifié dans un langage formel distinct), le M3 défini récursivement est une solution validée de manière *ad hoc*. Le méta-métamodèle MOF ne présente pas les caractéristiques d'un DSL d'expression de langage et il apparaît légitime de s'interroger sur la pertinence de l'approche, qui cependant semble en pratique bénéficier d'un consensus d'adoption.

Définition 3. *Méta-métamodèle : Un méta-métamodèle est un modèle qui décrit un langage de métamodélisation, c.-à-d. les éléments de modélisation nécessaires à la définition des langages de modélisation. Il possède de plus la capacité de se décrire lui-même.*

Le langage de métamodélisation, également nommé méta-métamodèle, est réflexif. Il y a donc quatre niveaux de hiérarchie dans la décomposition allant de M0 (le sujet) à M3 (le méta-métamodèle). Les modèles représentant cette réalité constituent le niveau M1. Les métamodèles permettant la définition de ces modèles (par ex. UML) constituent le niveau M2. L'organisation de cette modélisation est généralement décrite sous une forme pyramidale (cf. figure 2.3). Le langage de métamodélisation MOF 2.0 [OMG11a], est un standard créé par l'OMG, ECore en est un autre, lui *de facto*, initié par EMF [SBPM09, 2].

Un méta-métamodèle détermine le paradigme utilisé dans les modèles qui sont construits à partir de lui [Mul06]. L'approche qui consiste à considérer une hiérarchie de métamodèles n'est pas propre à l'IDM, et apparaît dans de nombreux domaines de l'informatique. Chaque hiérarchie définit un espace technique [KBA02, BK05]. Se distinguent par exemple le *modelware* (espace technique des modèles), les DSL Tools de Microsoft [GSCK04], le Generic Modeling Enviroment (GME) [LMB+01]), le *grammarware* [KLV05] (espace technique des grammaires définies par les langages tels que BNF[2]), le *BDware* (espace technique des bases de données), les ontologies (OWL), etc.. Une adaptation de la figure 2.3 pour le *grammaware* positionne l'EBNF au niveau M3, la grammaire au niveau M2 et les programmes au M1.

[2] Backus-Naur form

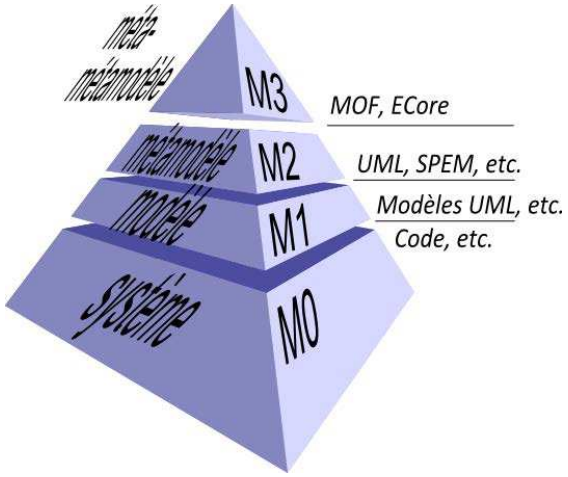


Figure 2.3: Pyramide de modélisation de l'OMG [OMG11a].

Définition 4. *Espace technique : Un espace technique est l'ensemble des outils et techniques issus d'une pyramide de métamodèles dont le sommet est occupé par une famille de (méta)métamodèles similaires [FEBF06].*

Dans l'espace technique de modélisation, dans une approche de type MOF, un métamodèle peut être appréhendé tel un uplet d'ensembles d'entités cognitives ($ECog$) et coercitives ($ECoe$), ces deux ensembles étant liés par une fonction associative. Les entités cognitives sont formées par les concepts et propriétés attachées (pragmatiquement des attributs et opérations). Les entités coercitives reposent sur des contraintes de bonne formation présentant une dualité : (i) des contraintes structurelles, imposées par diverses formes d'association entre les concepts (par ex. généralisation, référence, agrégation) ; (ii) des contraintes rattachées au métamodèle (par ex. par un langage distinct tel que OCL [OMG12]).

2.3.3 Des relations de l'espace de modélisation

La réalité des applications, chaînes d'outillages, *lignes de produits*, écosystèmes logiciels font naître un besoin accru de capture, non seulement des modèles, mais également de leurs relations et interactions. Tout d'abord, la relation présentée entre le sujet et le modèle est nommée *représentation de* - μ - (ou *RepresentationOf* en anglais). Ensuite la notion de métamodèle conduit à l'identification d'une seconde relation, liant le modèle et le langage utilisé pour le construire, appelée *conforme à* et nommée χ.

Des travaux s'intéressent spécifiquement aux relations entre les modèles. Hebig *et al.* [HSG11] proposent une synthèse de ces approches en les comparant, en termes de portée, définition, etc. Certains nomment ces approches *mégamodèles* ([BJV04, BDFB07] : *"A megamodel is a model with other models as elements"*; [Fav05b] : *"[...] the idea behind a megamodel is to define the set of entities and relations that are necessary to model some aspect about MDE"* (traductions p. 204), d'autres *macromodèles* (*"A macromodel consists of elements denoting models and links denoting intended relationships between these models with their internal details abstracted away"* [SME09] - traduction p. 204). Certaines approches tendent vers une gestion de traçabilité duale de bas et haut niveau [BDFB07, SNG10].

Pour résumer, Bézivin [BJV04, BDFB07] identifie deux relations fondamentales, nommément *RepresentationOf* (représentation de) et *ConformantTo* (conforme à). Favre montre quant à lui dans [Fav05b] que la relation *ConformantTo* correspond en réalité à un raccourci d'une combinaison des relations *RepresentationOf* et *ElementOf* - élément de - ($\chi \triangleq \mu\epsilon$). Dans l'approche linguistique des mégamodèles de Favre [Fav05b, Fav05a, Fav06], tout artefact IDM peut être décrit à partir de quatre relations basiques (plus une dérivée) qui sont : *RepresentationOf*, *ElementOf*, *DecomposedIn* (décomposé en), *IsTransformedIn* (est transformé en), et la dérivée *ConformsTo*.

La figure 2.4 illustre trois des quatre relations de Favre. La France fait office de sujet et le modèle qui en est réalisé (descriptif) est la carte (relation μ entre les deux éléments). En cartographie il est indispensable d'associer à chaque carte une description du langage (légende explicite). La carte doit être conforme à cette légende (relation χ), et plusieurs cartes peuvent être conformes à une même légende (relation élément de ϵ et μ).

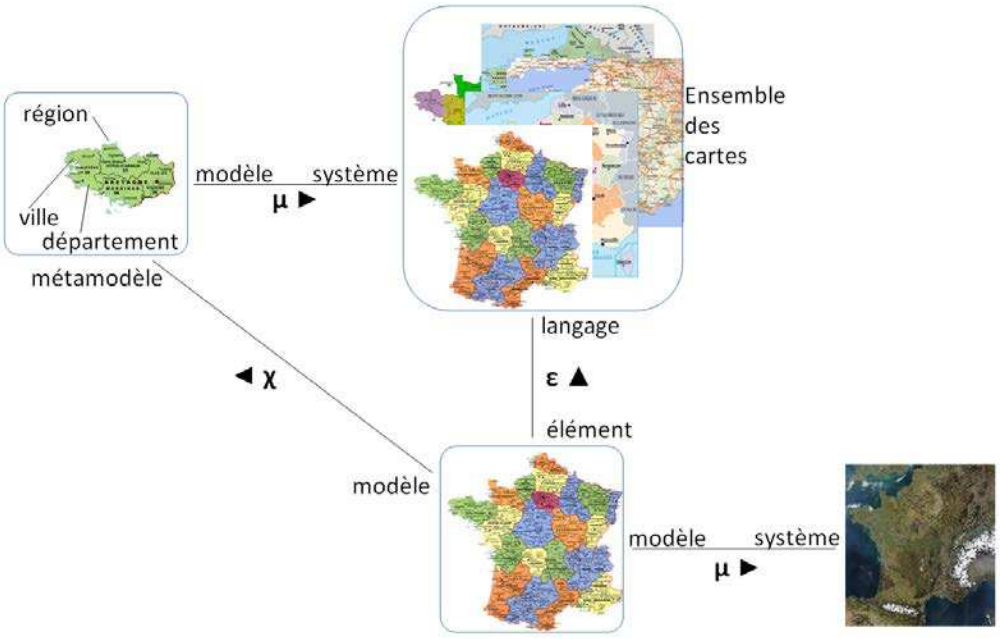


FIGURE 2.4: Relations entre système, modèle, métamodèle et langage [FEBF06].

Les travaux de Favre trouvent une continuité dans des applications du domaine des interfaces utilisateur [SCFC09], de la modélisation exécutable [VSG11], des espaces techniques [FLV12] et la modélisation ontologique [GKH07].

2.3.4 Modélisation intentionnelle

Muller *et al.* [MFBC10] focalisent leur étude sur la relation *RepresentationOf*, noté μ entre deux éléments (par exemple X et Y, cf. figure 2.5). Les auteurs construisent un langage de syntaxe diagrammatique basé sur des "choses" *("things")* et des "flèches" *("arrows")*.

Ni les choses ni les représentations de ces choses ne sont construites isolément, toutes deux existent pour un but précis, exhibent des propriétés, et sont élaborées pour des parties-prenantes données. Muller *et al.* prennent le parti de qualifier les relations entre "choses" par une intention.

La modélisation intentionnelle [YM94] répond aux questions telles que *qui* et *pourquoi*, mais pas *quoi*. L'intention d'une chose représente le motif d'utilisation par une personne de cette "chose", son contexte, et les attentes que l'on peut avoir de cette "chose". Le caractère intentionnel des modèles se retrouve également dans les travaux de Kuehne [Küh06] sur la métamodélisation, et dans ceux de Gašević *et al.* [GKH07] (une extension du mégamodèle de Favre) mais dans une moindre mesure.

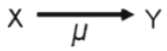


Figure 2.5: X est une représentation de Y[MFBC10].

Par la suite, la notation μ_I est utilisée, signifiant que X est une représentation de Y suivant une intention donnée.

Opposition entre les représentations analytique et synthétique

Tel qu'évoqué précédemment (cf. section 2.1.2), divers auteurs font une distinction entre les modèles analytiques (également nommés descriptifs par Favre [Fav05b]) et synthétiques (ou modèles de spécification [Fav05b]).

Une relation de représentation analytique spécifie que la source exprime quelque chose sur la cible ; la relation analytique est identifiée par μ_α et est définie comme :

$$x \rightarrow_{\mu_\alpha} y : \exists T_\alpha \mid x = T_\alpha(R(y))$$

dans laquelle T_α est une relation telle que X peut être dérivée (ou abstraite) de R(Y), et R étant une représentation de X (Identité incluse). Dans une approche dirigée modèles, T_α peut exprimer une transformation de modèles. En terme de vérité, cette dernière est portée par la cible dans le cas d'une représentation μ_α .

Une relation de représentation synthétique exprime que la cible est générée à partir de la source ; la relation synthétique μ_γ est définie comme :

$$x \rightarrow_{\mu_\gamma} y : \exists T_\gamma \mid y = T_\gamma(R(y))$$

dans laquelle T_γ est une relation telle que Y peut être dérivée (ou générée) à partir de R(Y), et R étant une représentation de X (Identité incluse). Dans une approche dirigée modèles, T_γ peut également exprimer une transformation de modèles. En terme de vérité, celle-ci est portée par la source dans le cas d'une représentation μ_γ. Dans le volet intentionnel, cela signifie que l'intention de Y peut être dérivée (synthétisée) depuis l'intention de X, ou du moins dirigée par l'intention de X (Y étant le résultat de T_γ appliqué à X).

Le tableau récapitulatif de la figure 2.6 issue de [MFBC10] présente les cinq sortes de μ-relations et leur notation graphique associée (figure traduite en annexe A, page 204). Le "type" est : différent, partagé, sous, sur, ou identique. La causalité et la transitivité des relations sont discutées dans [Sei03]. Les axiomes de composition des intentions suivent les règles définies par Muller *et al.* [MFBC10], et sont représentés dans la figure B.1 de l'annexe B (section B.1).

La figure 2.7 quant à elle propose une illustration des relations inspirée de la peinture de Magritte, extrait de la série "La Trahison des images" (1929, huile sur toile, 62 x 81 cm, Art Institute of Chicago). L'information contenue par l'image est réalisée avec une certaine intention en tête. La distribution de la couleur avec une toute autre intention, symbolisée par la flèche en pointillé.

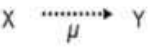
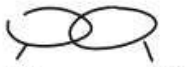
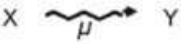
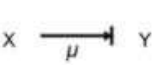
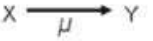
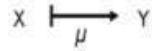
Kind	Intention	Description	Notation
different		X and Y have totally different intentions. This usually denotes a shift in viewpoints.	
share		X and Y share some intention. X and Y can be partially represented by each other. The representation is both partial and extended.	
sub		The intention of X is a part of Y's intention. Everything which holds for X makes sense in the context of Y. Y can be partially represented by X.	
same		X and Y share the same intention. They can represent each other. This usually denotes a shift in linguistic conformance.	
super		X covers the intention of Y; X can represent Y, but X has additional properties. It is an extended representation.	

FIGURE 2.6: Variations des μ-relations et notation graphique associée[MFBC10].

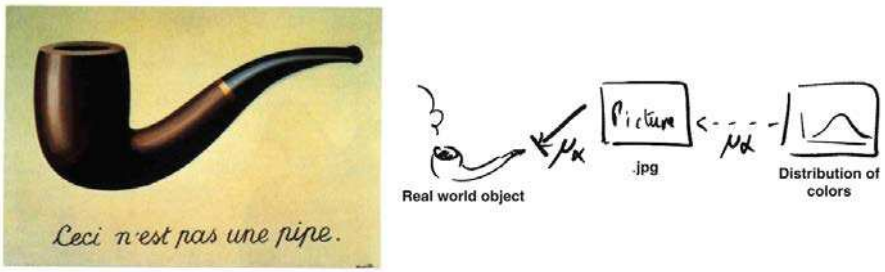


FIGURE 2.7: Exemple de relations μ_α [MFBC10], extrait d'une peinture de Magritte.

2.4 Des activités de l'IDM

Le principe fondateur de l'IDM est "tout est modèle". Suivant cette approche, tous les artefacts utilisés dans un cycle de développement produit (par ex. exigence, spécification, documents d'analyse et de conception, suite de tests et code) sont vus comme des modèles. Suivant ce point de vue modèle centré, les techniques et outillages permettant de manipuler ces modèles sont des *transformations de modèles*. Selon [SK03, GLR+02], la transformation de modèles est "le cœur et l'âme des développements dirigés par les modèles". Kleppe *et al.* [KWB03] fournissent la définition générale suivante :

> "[...] *a* transformation *is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of* transformation rules *that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language."*
>
> Kleppe *et al.* [KWB03] (traduction p. 204)

Cette définition pose la notion de transformation et de règles et doit être étendue pour inclure les transformations prenant plusieurs intrants et produisant de multiples modèles en sortie, à l'exemple du tissage et de la composition de modèles [FBFG07, HHKN09]. Mens et Van Corp [MVG06] proposent une taxonomie des transformations de modèles. Une étude de l'état de l'art (par ex. [CH06, Men10]) permet d'appréhender les différents concepts des transformations de modèles. Cette section relate quelques qualificatifs, une description exhaustive ne fait nullement l'objet de cette thèse. Se distinguent les transformations *endogènes* ou *exogènes* suivant que les modèles impliqués sont définis par le même langage ou non. La séparation des transformations en *verticales* et *horizontales* distingue (i) les modèles d'entrée et sortie à des niveaux d'abstraction distincts (exemple : génération de code), (ii) des modèles au même niveau d'abstraction (exemple : "refactoring"). En outre, le langage de transformation fait aussi l'objet de classification entre les approches (i) déclaratives (exemple implémentation de QVT Relation [8]), (ii) impératives (exemple Kermeta [18]), ou (iii) hybrides (exemple ATL [JABK08]).

La composition de modèles est un cas particulier des transformations permettant la mise en œuvre d'une séparation des préoccupations par modularité (cf. 2.2). En dépit d'une terminologie clairement définie, la littérature s'accorde globalement sur la classification de trois types d'activité d'intégration [CNS11] :

1. La fusion *(merge)*, basée sur la relation de chevauchement entre modèles *(overlap)* ; classiquement, les différents modèles représentent diverses perspectives d'une même fonctionnalité qui sont unies dans un nouveau modèle.
2. La composition *(compose)*, basée sur la relation d'interaction entre les modèles ; elle fait référence au processus d'assemblage d'un ensemble de modèles autonomes mais interagissant dans un même système.
3. Le tissage *(weave)*, basé sur la relation de "*cross-cut*" entre les modèles ; la relation est asymétrique, le résultat de différents modèles d'aspect est tissé sur le modèle de base *(Aspect-Oriented Modeling - AOM)*.

Notons que la composition désigne parfois l'ensemble de ces techniques. Une étude récente de Clavreul [Cla11] traite de la composition de modèles et de métamodèles. Dans les cas d'applications de la présente thèse, l'approche Kompose [FBFG07] et l'identification à base de signature [RFG+05] sont mises en application.

La génération d'outils à partir d'un métamodèle n'est pas nouvelle, Park *et al.* [PK93] proposent dès 1993 de générer l'outil de modélisation lui même, marquant une rupture avec les outils de type CASE. Ces idées ont fait leur chemin jusqu'à l'implémentation en Java, dans Eclipse de l'*Eclipse Modeling Framework* (EMF) [SBPM09, 2], et des projets associés (GMF, EMFText, etc.). Des outilleurs, tel Obeo, proposent des outils autour de cette plate-forme (par exemple Obeo Designer [11, JB10] pour la gestion de syntaxe concrète graphique).

2.5 Traçabilité et IDM

La traçabilité des artefacts logiciels est reconnue comme un facteur significatif de soutien aux nombreuses activités du processus de développement des systèmes logiciels [Poh96b]. Plus précisément, les informations de traçabilité peuvent être mises à profit dans l'analyse d'implication et d'intégration de changements d'exigences lors de la maintenance et l'évolution de systèmes. La traçabilité est une aide précieuse à la découverte de divergences et d'incohérences du point de vue des exigences en identifiant des connections entre des exigences disparates.

Les recherches en traçabilité logicielle ont plus particulièrement porté sur (i) l'étude et la définition de différents types de relations de traçabilité, (ii) le développement d'architecture, d'outillage, et d'environnements pour la représentation et la maintenance des relations de traçabilité, et sur (iii) des investigations empiriques de mise en œuvre d'approches concernant l'établissement et le déploiement de relations de traçabilité dans des cycles de développement logiciel.

D'autre part, Ramesh et Jarke relèvent [RJ01] que, classiquement, les relations de traçabilité dénotent (i) des situations de superposition (*"overlap"*), de satisfiabilité, de dépendance, d'évolution, de généralisation/raffinement, de conflit et d'explicitation de la logique, par des associations entre divers artefacts (par ex. exigences, spécifications, analyse logiciel, conception, modèles de test, codes source), ou (ii) des relations de contribution entre ces artefacts logiciels et des parties-prenantes ayant contribué à leur élaboration.

En fonction de leur sémantique, les relations de traçabilité exposent des informations pouvant être traitées de différentes manières dans le processus de développement logiciel et de nombreuses approches ont été proposées concernant la traçabilité logicielle ; les orientations scientifiques concernent plus particulièrement quatre axes distincts [SZ04], à savoir :

1. l'étude et la définition de différents types de relations de traçabilité, par ex. [CAS03, Dic02, EG02, GF95, Kai93, Let02, LS96, Poh96a, SZPMK04, DP05] ;
2. la fourniture de support à leur génération, par ex. [Ale02, EG02, PG96, SZPMK04] ;
3. le développement d'architecture, outillage et environnements pour la représentation et maintenance des relations de traçabilité, par ex. [PG96] ;
4. des investigations empiriques de mise en œuvre d'approches concernant l'établissement et le déploiement de relations de traçabilité dans des cycles de développement logiciel, par ex. [RJ01, AMR02].

Spanoudakis et Zismann [SZ04] mettent en exergue l'incohérence entre les efforts de recherche réalisés et la mise en pratique effective dans l'industrie des mécanismes de traçabilité. Toujours d'après les auteurs, ce phénomène peut être attribué à la difficulté d'automatiser la génération de relations de traçabilité ayant une sémantique claire et précise, permettant à moindre coût de délivrer des analyses apportant des bénéfices.

2.5.1 Définition

Le dictionnaire IEEE Standard Computer [oEE90] définit la traçabilité comme :

> - *"The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another* [...].
> - *The degree to which each element in a software development product establishes its reason for existing* [...]*."*
>
> IEEE Standard Computer [oEE90] (traduction p. 205)

Une autre définition de Gotel et Finkelstein [GF94] pose la traçabilité comme la capacité à décrire et à suivre la vie d'une exigence, dans deux directions, avant et arrière ; c.-à-d. depuis ses origines, tout au long de ses spécifications et de son développement, vers son déploiement et utilisation qui suivent.

Ces définitions sont orientées point de vue des exigences, qui est la discipline historique d'apparition de la traçabilité. La traçabilité couvre un plus large spectre dans les approches IDM, reliant tout type d'artefact de modélisation [ARNRSG06b], à tout type de niveaux d'abstraction.

Des "types" de relations de traçabilité

Des parties-prenantes avec diverses perspectives, différents buts et intérêts sont impliquées dans le processus de développement logiciel. Elles sont susceptibles de contribuer à la capture et la manipulation d'information de traçabilité.

Tout d'abord, un élément important, Dick défend dans [Dic02] que typiquement dans l'industrie, la sémantique de relations de traçabilité est très pauvre et qu'il est nécessaire de représenter des relations de traçabilité avec une sémantique plus riche et approfondie. Ce point de vue est également soutenu dans [PG96] et [BW02] ; dans le cadre des LdPs, ces derniers auteurs insistent sur le fait que les relations doivent avoir une signification sémantique plus riche au lieu de se cantonner à être de simples relations bi-directionnelles de référencement.

De nombreuses approches traitent ce problème pour établir des formes significatives de sémantique pour les relations. Des classifications de relations ont été établies, certaines basées sur le type d'artefacts connectés, par ex. [Egy03, LS96, Poh96b, ZSPmK03], d'autres reposent sur les scenarii et usages des relations [RJ01, Dic02, GF95, WJSA06, OO07] (compréhension, capture, suivi, évolution et vérification), ou encore sur les applications métiers [RPG07]. Paige *et al.* centrent leur étude sur le processus de réalisation d'une traçabilité [POKZ08].

En outre, Lindvall et Sandhal [LS96] classent les relations en *verticales* et *horizontales* suivant que ces relations ont pour extrémité des éléments d'un même artefact ou d'artefacts distincts. Une distinction similairement nommée est définie par [BBH+96] qualifiant des relations entre des modèles d'une même étape du processus de développement (*horizontale*), ou entre des modèles correspondant à différentes étapes (*verticale*). Cette dernière classification, renforcée par l'exploitation de niveaux d'abstractions, est prédominante dans la communauté IDM.

Une tout autre distinction est réalisée par [GF94] qui évoque la notion de relations de *pré-traçabilité* (entre les parties-prenantes ayant exprimé un besoin se traduisant en une exigence de spécification) et *post-traçabilité* (entre les exigences et les autres artefacts de conception créés lors du cycle de développement du logiciel).

Enfin, tel que consigné par [SZ04], les relations peuvent se grouper en :

Relation de Dépendance. Un élément e_1 *dépend* d'un élément e_2 si l'existence de e_1 repose sur l'existence de e_2, ou si un changement de e_2 se doit d'être répercuté sur e_1. Brièvement,

pour ne lister que quelques synonymes, les relations de dépendance se retrouvent dans [ZSPmK03, SZPMK04] sous l'appellation "*requires-features-in*", dans [MMMN03] "*causal conformance*", enfin dans [GF95] "*developmental relations*".

Relations de généralisation/raffinement. Elles tracent généralement la décomposition de système complexe. Egalement nommées "*abstraction links*" [Poh96b], ces relations sont utilisées pour représenter des entités à différents niveaux d'abstraction [KPKH02].

Relation d'évolution. Un élément $e1$ *évolue vers* un élément e_2 si e_1 a été remplacé par e_2 durant le développement, la maintenance ou l'évolution du système. Ce type de relation est spécialisé dans [Poh96b] en "*replace, based_on, formalize, elaborate*" et est qualifié de *non causal* dans [MMMN03] ou de *temporel* dans [GF95].

Relations de satisfiabilité. Un élément e_1 *satisfait* un élément e_2 si e_1 comble les attentes, besoins et intentions de e_2, ou s'il respecte une condition exprimée par e_1. Ces relations sont utilisées pour exprimer la satisfaction des exigences par le système solution.

Relations de recouvrement ou superposition. Un élément e_1 *se superpose à* un élément e_2 si e_1 et e_2 réfèrent à une caractéristique commune d'un système ou d'un domaine. Ces relations sont nommées "*representation*" dans [KPKH02]. Dans une approche de composition, ces relations spécifient l'activité de fusion *(merge)* des modèles.

Relations de conflit. Elles signifient des conflits entre deux éléments e_1 et e_2. De nombreux travaux traitent de ces relations (cf. [SZ04]). Dans le courant "gestion de la cohérence" du génie logiciel, par ex. [KZ02], les conflits sont fréquemment représentés par des relations d'incohérence ("*inconsistency*").

Relation d'explicitation du raisonnement. Ce type de relations est employé pour représenter et maintenir une logique de création et évolution d'éléments, de décision sur le système à divers niveaux de granularité. Il peut s'agir par exemple de conserver un historique d'actions réalisées [RJ01] ou d'expliquer les spécifications en regard des exigences [Let02].

Relation de contribution. Ce type de relations est consacré à la représentation d'associations entre des artefacts d'exigences et des parties-prenantes qui *contribuent* à la création de ces exigences, par ex. [GF95].

Applications de la traçabilité et outillage

Les relations de traçabilité sont déployées tout au long du cycle de développement d'un système logiciel comme fondement de différentes activités de développement et de maintenance, entre autre : (i) analyse et gestion des impacts face au changement [CHCS+02], (ii) vérification, validation de système, (iii) réutilisation d'artefacts logiciel [RJ01], et (iv) compréhension des artefacts logiciels (au niveau code [ACC+02], exigences [RD92, RJ01].

De l'outillage pour soutenir la gestion de la traçabilité a été développé. Parmi ces outils, certains sont relativement bien déployés dans le monde industriel, Reqtify [17], RequisitePro [5] et Obeo Traceability [12]. Une approche de génération de liens de traçabilité est présentée dans [Egy03] avec un outillage de dérivation automatique des traces depuis le code vers les exigences.

L'ouvrage [CHGZ12] fournit un état de l'art récent sur la traçabilité pour les systèmes et logiciels.

2.5.2 La traçabilité pour les modèles

Il existe un certain nombre de travaux autour de la modélisation et de la traçabilité, et des auteurs tels [GG07, WvP10] ont réalisé des études sur les approches de traçabilité dans les développements IDM. Certaines approches se centrent autour des exigences (par ex. [CHCS+02, RJ01]), d'autres redéfinissent des métamodèles *ad-hoc* (par ex. [Egy03, OOK10, POON10]), ou enfin définissent des traçabilités de transformations [Jou05, rFHN06].

Les usages de la traçabilité sont divers, et il est possible de relever, entre autres, le support des décisions de conception ([BR08]), la validation d'artefacts (détection d'incohérences, d'incomplétudes, etc. [RE93]), l'amélioration de la modifiabilité (par ex. [GF94, RE93]), de la compréhension du système (à partir de points de vue [SE05], l'unification d'informations fragmentaires [GF94], l'amélioration du processus [ARNRSG06a, POKZ08]), la création d'éléments dans une ligne de produits existante [AR05]) et le contrôle du développement des LdPs (par ex. [AKM+10]).

Précédemment évoquées dans la section 2.3.3, des approches [BDFB07, SNG10] tendent à rapprocher les mégamodèles et le domaine de la traçabilité. Une dualité de niveaux de traçabilité est introduite dans des processus IDM, déterminant un niveau local (liens de traçabilité de transformations) et un niveau global (mégamodèle) de la traçabilité pour associer tous les artefacts IDM.

2.6 Discussion et synthèse

Le chapitre présente le cadre théorique de cette thèse et les concepts clés (en réponse à **QR 1**) pour la compréhension de l'étude. La thèse s'inscrit dans le domaine de l'Ingénierie Dirigée par les Modèles, où les notions d'abstraction et de séparation des préoccupations sont de premier ordre. L'importance de l'aspect relationnel est mise en valeur par les approches de traçabilité et de gestion de modèle (modélisation de la modélisation). Dans le paradigme théorique général, la notion d'intention ressort comme étant un élément clé.

2.6.1 Des bénéfices de l'approche IDM

D'après [OS00, OMG03b], les bénéfices spécifiques aux approches dirigées par les modèles sont (i) la productivité, (ii) la réduction des coûts de développement (automatisation de génération de code et de test), (iii) l'amélioration de la portabilité (standardisation et formalisme, niveaux d'abstraction et format d'échange commun), (iv) la réduction du temps de développement, (v) l'amélioration de la qualité (des modèles par simulations et tests dans les phases amonts, du code généré, par une détection de bugs gérée au plus tôt), (vi) la facilité de maintenance et d'évolution (obsolescence des plate-formes), (vii) l'amélioration de la communication et du partage des informations (entre les parties-prenantes et les membres de l'équipe). Dans l'ensemble, la principale raison économique derrière l'approche dirigée par les modèles concerne un gain de productivité, ce qui est rapporté dans des études [OMG03b, 13].

2.6.2 De l'importance de l'étude des relations et de leur sémantique

Face aux différents défis de la recherche dans le domaine de la traçabilité, des travaux collaboratifs ont émergé dans la communauté pour aboutir à la définition des grands défis *(Grand Challenges of Traceability)* [GCHH+12]. Selon ce rapport, la traçabilité doit être adaptée à un objectif donné, afin de soutenir les besoins des parties-prenantes (*Purposed Goal 3*, [GCHH+12]). Certaines exigences peuvent être relevées, comme la suivante : le besoin de comprendre le paradigme employé pour développer le système logiciel (par exemple, orienté objet, agent ou service, *ligne de produits*), la nature des artefacts impliqués et la contextualisation de la traçabilité dans le processus métier (*Purposed Req 12*, [GCHH+12]).

Un objectif de configurabilité propose une traçabilité planifiée dans un projet, utilisant une sémantique riche des liens de traçabilité (*Configurable G1*, [GCHH+12]). Les relations ne peuvent pas être décorrélées de leurs usages.

Une des exigences relevées concerne la mise à l'échelle et le besoin de spécifier le concept de granularité. Les approches existantes définissent des niveaux macro et micro de granularité sans réel consensus sur la signification de grain fin *(fine-grain)* et de gros grain *(coarse-grain)*.

Pour conclure, il est important de relever la recommandation suivante :

R 1. *Une modélisation relationnelle doit être spécifique à un domaine métier et à son processus de développement et de contenu sémantique riche.*

2.6.3 De l'importance de l'intention

Ce besoin d'expliciter les intentions entre en résonance avec plusieurs théories en sciences humaines, comme en premier lieu la théorie de l'activité [Leo74, Eng87] qui intègre la composante intentionnelle dans la description de l'activité, ou la théorie de la genèse instrumentale et de la conception continuée dans l'usage [Rab95].

Des travaux spécifiques dans le champ du génie logiciel s'intéressent à l'ingénierie des besoins orientée par les buts [LL04, Yu97]. Ces travaux considèrent l'explicitation des buts ou des intentions comme un point de départ à la spécification des systèmes d'information. La notion d'intention prend tout son sens dans le domaine des processus [YM94]. Rolland [Rol98] décrit un modèle de processus exprimé selon une perspective orientée par les buts : "Un but définit une intention de réaliser un processus dans le cadre d'un métier (avec ou sans système)". L'approche ne se concentre pas uniquement sur l'expression des intentions (ou buts) mais également sur les différentes manières de les réaliser (les stratégies). Dans une approche basée sur SPEM [OMG08], Koudri *et al.* [KC10] réalisent une dichotomie entre l'intention et la méthodologie.

Or à ce jour, seul les travaux de Muller *et al.* [MFBC10] proposent un cadre générique de modélisation des relations intentionnelles. Au vu de l'importance de l'explicitation de l'intention, le besoin suivant se fait ressentir :

B 1. *Une modélisation de modèles impliqués dans un processus de développement nécessite une explicitation de l'intention.*

CHAPITRE 3

Modélisation de la variabilité

"Handle stets so, daß die Anzahl der Wahlmöglichkeiten größer wird." [a]

[a] Agis toujours de façon à augmenter le nombre de choix possibles.

— Heinz von Foerster, *Über das Konstruieren von Möglichkeiten, 1973*

Un besoin grandissant de variabilité s'est fait ressentir dans les développements de systèmes logiciels dans les dernières décennies, c.-à.-d. une demande d'habileté des artefacts logiciel à varier dans leur comportement à un instant donné dans leur cycle de vie.

Il existe deux explications sous-jacentes à ce phénomène. De prime abord, notons que la variabilité des systèmes est translatée de la mécanique et du matériel vers le logiciel. Ensuite, en raison des coûts engendrés par un changement de décision de conception, les ingénieurs échelonnent certaines décisions en les programmant dans les dernières phases de développement (économiquement défendable selon [HLSS98]). L'approche IDM propose en cela une résolution implicite de la variabilité en échelonnant les décisions de conception à plusieurs niveaux d'abstraction.

La seconde tendance tend à expliciter la variabilité. Elle est prépondérante dans l'approche de réutilisation dite des *lignes de produits* [WL99, CN01]. La réutilisation est un sempiternel dessein de l'ingénierie logicielle, marqué par McIlroy [McI68] et son célèbre discours à la conférence de l'OTAN où le terme de composant logiciel a été introduit. Par la suite, Parnas [Par76] présente la notion de famille de programmes et, implicitement, montre comment des modules logiciels sont réutilisés entre les produits (cf. citation ci-après). Il expose également une manière de retarder la prise de décision de sélection des modules logiciels, introduisant par là implicitement la notion de point de variation.

> *"We consider a set of programs a family if they have so much in common that it pays to look at their common aspects before looking at the aspects that differentiate them. This rather pragmatic definition does not tell us what pays, but it does explain the motivation for designing program families. We want to exploit the commonalities ... and reduce ... costs."*
>
> David Lorge Parnas, 1979 (traduction p. 205)

Après avoir abordé la notion de modèle, d'abstraction et de préoccupations, ce chapitre se focalise sur une préoccupation particulière qu'est la variabilité et sa modélisation. La présentation détaillée de la *ligne de produits* et de son processus seront explicités dans le chapitre suivant (chapitre 4). Le présent chapitre s'intéresse à la définition de la variabilité (section 3.1), des différents modèles de variabilité existants (section 3.2), et ils sont pléthores, pour détailler le modèle OVM puis le modèle de *features* (section 3.3). La dernière section présente une étude des relations dans les modèles de variabilité, créées pour réassocier des préoccupations diverses (section 3.4).

Ce chapitre répond partiellement à la question de recherche suivante :

QR 2. *Quels sont les artefacts manipulés ?*

3.1 La variabilité

Un parcours de la littérature des *lignes de produits* révèle de nombreuses définitions de la variabilité dont est présentée une sélection ci-après. La variabilité est liée à la LdP, car définie dans son contour, localisant les différences entre les produits d'une même famille. Les approches portent chronologiquement d'avantage leur attention sur la résolution de cette variabilité. Le pendant de cette variabilité est la *commonalité*, qui regroupe l'ensemble des hypothèses vérifiées pour tous les produits de la famille [WL99].

Il se distingue deux courants dans les approches, celui modélisant la variabilité, qui prend en considération la variabilité et la commonalité, et celui modélisant les décisions de résolution de la variabilité, faisant fi de la commonalité.

3.1.1 Définitions

La variabilité est corrélée à la capacité de modifier ou personnaliser un système. Une amélioration de la variabilité pour un système implique une amélioration de la facilité à réaliser certains types de changements. Il est envisageable d'identifier et d'anticiper certain type de variabilité et de construire le système d'une manière à accepter cette variabilité. Le tableau 3.1 regroupe quelques définitions de variabilité de la communauté des *lignes de produits*. Ces définitions font apparaître les notions d'élicitation et d'explicitation de la variabilité, ainsi que des solutions de résolution de cette dernière. La variabilité peut être définie à toutes les étapes du processus de développement, et dans une approche orientée modèles à différent niveaux d'abstraction.

Définition 5. Point de variation *: un point de variation identifie et localise l'endroit ou se produit la variabilité. Il précise les solutions possibles de résolution de cette variabilité (explicitation des* variants*), et éventuellement le moment de cette prise de décision (en anglais "binding time").*

Le corollaire de ceci est que le *variant* est une réalisation possible de la variabilité indiqué par le point de variation. La prise de décision et fixation de la variabilité peut intervenir à différents moments du cycle de vie du système, par ex. à la conception, à la compilation, à l'édition de liens, au déploiement, à l'exécution.

Van Gurp *et al.* [GBS01] distinguent trois états aux points de variation :

- Implicite : si la variabilité est introduite à un niveau d'abstraction donné, cela signifie qu'à un niveau d'abstraction supérieur cette variabilité est présente et par conséquence implicite.
- Conçu (*"designed"*) : dès que le point de variation est explicité, il est signifié comme conçu.
- Résolu (*"bound"*) : l'objectif premier du point de variation est de se lier à un variant appartenant à un ensemble de variants candidats à la résolution de la variabilité ; une fois réalisé, le point de variation est qualifié de résolu (ou fixé).

En outre, les auteurs définissent également les notions de points de variation ouverts et fermés ; ouvert si le point de variation est toujours susceptible d'accepter un nouveau variant, fermé dans le cas contraire.

Définition 6. Variabilité *: la variabilité est la description des variations possibles d'un système par des points de variation.*

Tableau 3.1: Définitions de la variabilité dans les Lignes de Produits.

[WL99]	"an assumption about how members of a family may differ from each other"
[BC05]	"variability means the ability of a core asset to adapt to usages in the different product contexts that are within the product line scope"
[PBL05]	"variability that is modelled to enable the development of customized applications by reusing predefined, adjustable artefacts"
[SvGB05]	"variability is the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context"

traductions p. 205

La gestion de la variabilité inclut les activités suivantes : élicitation et explicitation de la variabilité des artefacts tout au long du cycle de vie, gestion des dépendances entre ces différentes variabilités, support pour l'instanciation de ces variabilités. Une gestion de la variabilité de bonne facture est importante au succès de la *ligne de produits*. La suite de la section suivante (3.2) s'intéresse plus particulièrement à l'explicitation de cette variabilité dans des modèles.

3.1.2 Classification et types de variabilité

Des approches tendent à réaliser des distinctions entre des catégories de variabilité. Concernant le niveau des exigences, Halmans et Pohl [HP04] font une distinction entre la variabilité *essentielle* (*"essential"*) et *technique* (*"technical"*). La variabilité essentielle est celle dédiée au point de vue client, définissant et délimitant les artefacts à implémenter. La variabilité technique correspond quant à elle à une utilisation par les ingénieurs, définissant de ce fait, la manière d'implémenter les artefacts.

Lors de travaux plus récents, Pohl et al. [PBL05] généralisent ces notions à toute la *ligne de produits* et définissent la *variabilité externe* comme celle visible du client, et la *variabilité interne* comme celle du domaine, et qui par conséquent lui est masquée. Toujours dans le même ouvrage [PBL05], les auteurs mettent en exergue les dimensions de la variabilité : (i) *variabilité dans le temps* (différentes version d'un même artefact), (ii) *variabilité dans l'espace* (différents variants d'un artefact à un temps donné).

Concernant la variabilité dans le temps, Elsner *et al.* [EBLSP10] revisitent la variabilité en l'apposant orthogonalement à la variabilité dans l'espace. Les auteurs généralisent les usages qui en sont fait et font émerger trois types distincts de variabilité (dans le temps) :

- Une variabilité due aux changements linéaires dans le temps, c.-à-d. relative à la maintenance et à l'évolution ;
- Une variabilité ayant trait à différentes versions d'un artefact à un moment donné, c.-à-d. correspondant à de la gestion de configuration ;
- Une variabilité à résoudre dans le temps, c.-à-d. liée à la dérivation de produits dans un contexte de *ligne de produits*.

En outre, Deelstra *et al.* [DSB09] cherchent à évaluer l'évolution de la variabilité dans les LdPs logicielles. Dans leur approche, les auteurs distinguent (i) des modèles de variabilité fournie ("*provided* variability models"), c.-à-d. la variabilité reflétant les artefacts de la LdP, et (ii) des modèles de variabilité requise ("*required* variability models"), c.-à-d. la variabilité nécessaire à la réalisation des scenarii de dérivation des produits.

Savolainen *et al.* séparent, quand à eux, les propriétés *déductives* des *déclaratives* d'une famille de produits [SK01].

D'autre part, et dans la continuité des travaux de Pohl *et al.* [PBL05], Metzger *et al.* [MPH+07] proposent de scinder la variabilité en deux : (i) la variabilité de la ligne de produit ("*product line variability*"), et (ii) la variabilité du logiciel ("*software variability*"). La première réfère aux variations entre les systèmes appartenant à une LdP en termes de propriétés et qualités ; la seconde relève de l'habileté d'un système logiciel à être efficacement étendu, modifié, spécialisé ou configuré pour un besoin donné.

Enfin, Classen *et al.* s'attellent dans [CHS08] à préciser le sens d'un *feature*[1] pour le domaine de l'ingénierie des exigences. Les auteurs suivent la décomposition prônée par les travaux de Jackson et Zave [Jac95, ZJ97] et fournissent une définition en regard des exigences ("*requirement(R)*"), des spécifications ("*specification(S)*") et des hypothèses du domaine ("*domain assumptions(W)*"). Dans la même lignée, Tun *et al.* [TTBC+09] proposent une séparation des préoccupations de ces modèles de *features* selon le cadre de Jackson et Zave.

3.1.3 Granularité de la variabilité

Kircher *et al.* rapportent sur l'importance de séparer l'espace du problème de celui de la solution [KSG06]. Traitant des exigences, ces derniers insistent sur la séparation à réaliser entre des exigences clients et des exigences techniques de plus bas niveaux.

De leur côté, Kim *et al.* [KHC05] évoquent une variabilité conventionnelle et une variabilité des composants. Les auteurs présentent cinq types de variabilité : (i) des attributs, (ii) de logique (algorithmique), (ii) de flux ("workflow"), (iv) de persistante, et (v) d'interface. Le périmètre (*"scope"*) de la variabilité est également déterminé comme étant : binaire, de sélection, ou ouvert.

Etxeberria *et al.* [ES08] explicitent et distinguent les *features* fonctionnels de ceux de relatifs à la qualité. Néanmoins, Moreira *et al.* défendent, dans le domaine des exigences, que la séparation des préoccupations dans les modèles de *features* ne peut être restreinte à une bi-dimensionnalité fonctionnelle / non-fonctionnelle, les deux dimensions étant transverses [MRA05]. Les auteurs avancent une division en *meta concerns* (préoccupations génériques et communes à plusieurs systèmes) et *system concerns* (préoccupations spécifiques à un système donné).

3.2 Différents modèles de variabilité

La section offre une vue d'ensemble de la variabilité des modèles de variabilité et présente plus spécifiquement le modèle *OVM* [PBL05].

En raison de cette grande diversité de représentation, de nombreux auteurs présentent des études sur les moyens d'expression de la variabilité [TH03b, ASM04, SvGB05, HMPO05, SD07, MPFM08, CABA09, IKPJ11]. Chen *et al.* [CABA09] proposent une revue systématique des approches de modélisation de la variabilité. Leur étude fournit une liste chronologique des approches de modélisation sans apporter de classification. Mujtaba *et al.* [MPFM08] utilisent une méthode de *mapping* systématique afin de révéler des catégories et d'organiser la littérature existante selon celles-ci. Le schéma utilisé répartit les travaux suivant le contexte, le type de contribution, et le

[1] la section 3.3 présente en détails les approches de modélisation par *features*

type de recherche réalisé. L'article donne des indications sur les directions de recherche réalisées et à couvrir.

Dans un rapport technique, Asikainen *et al.* [ASM04] classent les méthodes selon qu'elles sont orientées *"feature"*, basées sur l'architecture, ou tout autre. Haugen *et al.* introduisent un modèle de référence pour comparer les approches. Ce modèle est basé sur une distinction de la sphère générique (*Feature Model* - FM, modèles de la LdP) et sur la sphère spécifique (sélection des *features* et modèles produits). Les catégories énoncées sont (i) l'utilisation d'un langage standard, (ii) l'annotation de langages génériques et (iii) l'utilisation d'un DSL dédié. Enfin, Istoan *et al.* fournissent une classification basée sur les métamodèles des approches de modélisation de la variabilité [IKPJ11]. Les auteurs différencient les méthodes utilisant un modèle unique pour représenter les artefacts de conception de la solution et le modèle de variabilité (les méthodes d'annotation et de fusion de DSLs avec un métamodèle de variabilité), et celle proposant un modèle de variabilité séparé (utilisation de FMs, de DSLs de variabilité orthogonaux, de modélisation de décision).

Les approches de modélisation des décisions[2] (*"decision modeling"*) ont pour caractéristique de manipuler (modélisation, documentations, ...) uniquement les propriétés ayant trait à la variabilité d'une famille de produits. Schmid *et al.* proposent, dans [SRG11], de comparer cinq approches et ainsi révéler leurs différences. Les approches existantes [SRG11] découlent pour une grande part de la "synthesis method" et définissent le modèle de décision comme :

> *"[...] a set of decisions that are adequate to distinguish among the members of an application engineering product family and to guide adaptation of application engineering work products."*
>
> Schmid *et al.* [SRG11] (traduction p. 205)

Comparativement aux approches de modélisation de la variabilité, du type modèles de *features* et autres, la modélisation des décisions met l'accent sur la dérivation du produit et non sur l'expression du domaine. Czarnecki *et al.* [CGR+12] font le lien entre ces deux catégories d'approches et proposent une comparaison systématique des FMs et des modèles de décision. Les auteurs concluent que les deux modèles partagent maintenant un même niveau d'expressivité.

La description ci-après s'inspire des travaux précédemment cités, mais ne tient pas compte de toutes les approches de modélisation de la décision, s'intéressant uniquement à celles d'entre elles proposant des intérêts du point de vue de l'expression de la variabilité.

3.2.1 Une représentation de la variabilité indépendante

La catégorie regroupe les méthodes dans lesquelles l'expression de la variabilité reste indépendante des artefacts de la *ligne de produits*. Ces approches sont pour la plupart anciennes, les approches plus récentes visent un automatisme et tentent d'assurer et d'optimiser les liens vers des artefacts de la solution.

Les méthodes basées documentation

Un outil, somme toute sommaire, reste la feuille Excel qui peut, ou a pu, parfois être utilisée pour représenter la variabilité.

[2] originellement introduit dans "synthesis method" [Con93]

Les méthodes basées "feature diagrams"

Ces méthodes mettent en œuvre des diagrammes de caractéristiques indépendants des artefacts de conception, à savoir le code, la documentation et les modèles. Ce type d'approche naît avec l'apparition de la méthode d'analyse FODA [KCH+90] et tend à disparaître au profit de méthodes davantage orientées vers l'implémentation. Les approches suivantes appartiennent à cette catégorie : FODA [KCH+90], FORM [KKL+98], FeatureRSEB [GFA98], Generative Programming [CE00], van Gurp *et al.* [GBS01], Riebisch *et al.* [RBSP02], Pluss [EBB05].

3.2.2 Une représentation de la variabilité couplée aux artefacts de la LdP

Tel que relevé par Haugen *et al.* [HMPO+08], il y a deux catégories de techniques qui introduisent la variabilité dans les langages de modélisation : une approche de fusion (*"amalgamated"*) et de séparation (*"separated"*).

La variabilité fusionnée

Ce type d'approche propose d'augmenter les capacités d'un langage par des concepts de variabilité. Ces langages peuvent être des DSMLs ou un langage générique de méta-modélisation, typiquement UML. La caractéristique distinctive est l'incorporation de la variabilité et des éléments de conception de la LdP dans un modèle unique.

Le modèle de base peut être annoté par une extension. Des auteurs étendent le métamodèle UML par des profiles dans des diagrammes tels que les diagrammes de Classes, de Séquences, de Use-Cases. Les travaux suivants sur le sujet sont identifiés : [ABB+02, Cla01, GS02, ZHJ03, Gom04, dOGHM05, ZJ06]. Concernant des DSMLs, Clafer [BCW11] est un langage incorporant une réification des concepts de modélisation par *features*. Morin *et al.* proposent dans [MPL+09, PVM+12] une approche prônant l'utilisation d'un métamodèle générique de variabilité et de son tissage dans n'importe quel DSML, au niveau métamodèle.

La variabilité séparée

Dans cette catégorie d'approches, une séparation de la préoccupation de modélisation de la variabilité est clairement établie, en effet, les métamodèles des artefacts de conception et du langage de variabilité sont indépendants.

Une majeure partie des approches utilise les modèles de *Features*, par exemple [CA05, HKW08, PKGJ08, LGB08, AJTK09]. Des fragments de modèles de conception sont associés aux *features* afin d'offrir une possibilité d'automatisation de la dérivation de produit ; les techniques pour "lier" les *features* aux artefacts de conception diffèrent d'une approche à l'autre.

D'autre DSLs spécifiques d'expression de la variabilité sont également mis en pratique. Pohl *et al.* défendent l'utilisation d'un langage de modélisation orthogonal de la variabilité nommé *OVM* [PBL05, MPH+07]. Le modèle de variabilité est relié aux artefacts de conception par des relations de dépendance ("artefact dependencies"[PBL05]). La méthode COVAMOF introduite par Sinnema *et al.* [SDNB04, SDH06, SDNB06] insiste sur une réification et uniformité de représentation des points de variation à tous les niveaux d'abstraction des artefacts de conception. L'approche propose une représentation hiérarchique des points de variations, une réification des dépendances et des interactions entre dépendances. VML* [ZSS+10] introduit une famille de langages textuels dédiée à la modélisation des relations entre éléments. Une comparaison des capacités de VML* et FeatureMapper en termes d'expressivité, automatisation et passage à l'échelle est réalisée dans [HSS+10].

Les approches de modélisation des décisions s'apparentent également à cette catégorie. Le modèle capture la variabilité en termes de résolutions possibles guidant la dérivation d'un produit. L'approche la plus connue de cette catégorie est DOPLER [DGR11].

Pour terminer, notons les travaux de standardisation d'un langage de variabilité, le "Common Variability Language (CVL)" [HMPO+08, FHMP+09, SZLT+10]. Ces travaux proposent un langage commun pour modéliser la variabilité de différents DSMLs, applicable à tout modèle (défini par un métamodèle conforme au MOF). Le langage manipule la notion de décision sans être pour autant dans la lignée et donc la sémantique des approches de modélisation des décisions de la littérature.

3.2.3 Implémentation de la variabilité

Diverses techniques d'implémentation de la variabilité ont été développées par la communauté, outillées et évaluées, au niveau de la gestion du code et au niveau modèle. Une étude du début des années 2000 [AG01] propose un panel de techniques de gestion de la variabilité au niveau implémentation, nous pouvons évoquer : les techniques de compilation conditionnelle, les techniques liées aux langages de programmation (par ex. classes templates de C++), les patrons de conception (par ex. la fabrique abstraite [Jéz99]), des approches de programmation spécifiques (par ex. programmation par aspects [RRR11], programmation générative [CE00, BLHM02]). Une taxonomie de ces techniques par Svahnberg *et al.* [SvGB05] distingue deux groupes d'approches : annotée et compositionnelle ; que nous retrouvons sous l'appellation variabilité négative et positive [VG07] au niveau des modèles. D'autre part, dans CVL, le langage contient un sous-ensemble définissant un langage d'action d'implémentation de la variabilité permettant la mise en œuvre de variabilité négative, positive et de la combinaison des deux (par ex. l'opérateur de substitution permet de retirer des artefacts et de les remplacer par d'autres). Des outils, tels que *Gears* [Kru07] ou *pure*::*variant* [Beu03, Beu08] permettent également un couplage de la variabilité avec plusieurs types d'artefacts : code, documentation, voire modèles.

Variabilité négative

Dans les approches de LdPs orientées modèle, Czarnecki *et al.* [CA05, CP06], Heidenriech *et al.* [HKW08, HSS+10], et dans les approches de *"model checking"* par ex. [LPT09], sont utilisées des annotations sur des éléments de modèles. Ces approches mettent en pratique un modèle complet de la LdP duquel sont supprimées des propriétés ou parties de modèle lors de la dérivation d'un produit. Les caractéristiques non désirables (non sélectionnées) lors de la configuration ne sont pas activées et conséquemment retirées pour constituer le produit final.

Variabilité positive

Les approches compositionnelles sont réalisées au travers de nombreuses techniques, par ex. : les "frameworks", les aspects [MO04], le raffinement itératif [BSR03]. L'idée de la variabilité positive est de composer des modèles ou fragments de modèles lors d'une phase d'intégration afin d'obtenir un produit répondant à une configuration donnée. Des approches de modélisation orientée tissage d'aspects ("AOM") ont été appliquées aux LdP, par ex. [MVL+08, AJTK09]. L'approche de Perrouin et al. [PKGJ08] propose une dérivation de produit par fusion de fragments de modèles.

3.2.4 OVM en détails

Cette section présente plus en détails, le modèle de variabilité orthogonale OVM [PBL05, BLP05, MPH+07]. En effet, ce modèle ainsi que le modèle de *features* (décrit dans la section suivante)

sont utilisés dans cette thèse.

OVM s'articule autour de la notion de point de variation (*variation point* VP) qui modélise les éléments variables. Un *variant* modélise les instances possibles d'éléments variables et est donc relié à un VP. Les VPs et variants peuvent être tous deux optionnels ou obligatoires ; un VP obligatoire doit toujours être sélectionné et par conséquent les variants qui lui sont attachés également. Un variant obligatoire doit être sélectionné *ssi* son VP est sélectionné. Les variants optionnels peuvent être groupés en choix d'alternative *(alternative choices)*, contraints par une cardinalité de type *min..max*. Des contraintes additionnelles entre VPs et entre VPs et variants peuvent être définies de manière graphique : *exclude* (exclusion mutuelle) et *requires* (implication). La figure 3.1 présente la syntaxe diagrammatique du langage [PBL05].

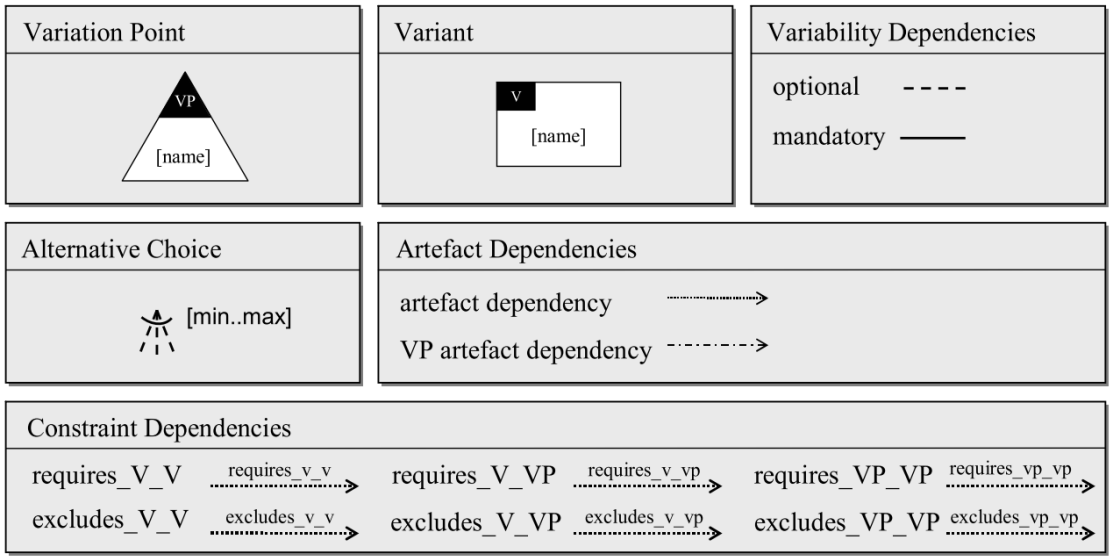


Figure 3.1: Syntaxe graphique du langage OVM [PBL05].

En terme de formalisation du langage, un modèle OVM Ω est considéré comme un 9-uplet de la forme[3] $\langle VP, V, VG, Parent, Min, Max, Opt, Req, Excl\rangle$ tel que :

- $VP(\neq \emptyset)$ est l'ensemble des points de variation VP ;
- $V(\neq \emptyset)$ est l'ensemble des variants ; $VP \cap V = \emptyset$;
- $VP(\neq \emptyset) \subset \mathcal{P}(V)$ est l'ensemble des groupes de variants ; VG partitionne V ;
- $Parent : V \cup VG \rightarrow VP$ est la fonction qui retourne le point de variation parent d'un variant donné ;
- $Min : VG \rightarrow \mathbb{N}$ et $Max : VG \rightarrow \mathbb{N} \cup \{*\}$ retourne la cardinalité d'un groupe de variant donné ;
- $Opt : V \rightarrow \mathbb{B}$ renseigne sur l'optionalité d'un Variant ;
- $Req \subseteq (V \times V) \cup (V \times VP) \cup (VP \times VP)$ représente les liens de contraintes d'implication ;
- $Excl \subseteq (V \times V) \cup (V \times VP) \cup (VP \times VP)$ représente les liens de contraintes d'exclusion mutuelle ;

Les règles de conformité additionnelle sont les suivantes :

[3] syntaxe abstraite issue de [MPH+07], avec une action corrective sur l'élément *Opt* pour se rapprocher de [PBL05]

- Tous les VPs sont parents d'au moins un V : $\forall vp \in VP \cdot \exists vg \in VG \cdot Parent(vg) = vp$ et $\forall vg \in VG \cdot vg \neq \emptyset$.
- Un VG et ses Vs ont le même parent : $\forall vg \in VG, v \in V \cdot v \in vg \Rightarrow Parent(v) = Parent(vg)$.
- Les cardinalités doivent être correctes : $\forall vg \in VG \cdot 0 \leq Min(vg) \leq Max(vg) \wedge 1 \leq Max(vg) \leq \sharp vg$.

À des fins d'illustration, le cas d'exemple *ItemCatalog SPL*, issu de la littérature [CP06, PKGJ08], est représenté dans le langage OVM. La figure 3.2 donne un aperçu du modèle dans la syntaxe graphique issue de [PBL05]. Le catalogue ("*Catalog*") d'une application de e-commerce propose des descriptions des articles ("*ProductInformation*") et optionnellement leur organisation en catégories ("*Categories*"). Chaque article peut être associé à une représentation 2D ("*2DImage*") ou 3D ("*3DImage*") ou les deux. Trois types de catégories sont disponibles permettant : une appartenance à plusieurs catégories distinctes ("*MultipleClassification*"), un support de description en sous-catégories ("*MultipleLevel*"), une fourniture de détails de description ("*Description*"), un aperçu du contenu ("*Thumbnails*"). La sélection de *Thumbnails* implique une nécessité de sélection de *2DImage* pour être cohérent.

La figure 3.2 présente une représentation du cas d'illustration avec la syntaxe graphique issue de [PBL05]. De manière abstraite, le modèle peut s'exprimer sous la forme :

- $VP = \{CS, AA, CT\}$;
- $V = \{PI, C, 2D, 3D, MC, ML, D, T\}$;
- $VG = \{2D3D\}$;
- $Parent(PI) = CS$; $Parent(C) = CS$; $Parent(2D) = AA$;$Parent(3D) = AA$; $Parent(2D3D) = AA$; $Parent(MC) = CT$; $Parent(ML) = CT$; $Parent(D) = CT$; $Parent(T) = CT$;
- $Min(2D3D) = 1$; $Max(2D3D) = 2$;
- $Opt(PI) = False$; $Opt(C) = True$; $Opt(2D) = False$; $Opt(3D) = False$; $Opt(MC) = True$; $Opt(ML) = True$; $Opt(D) = True$; $Opt(T) = True$;
- $Req = \{(PI, AA), (C, CT), (T, 2D)\}$;

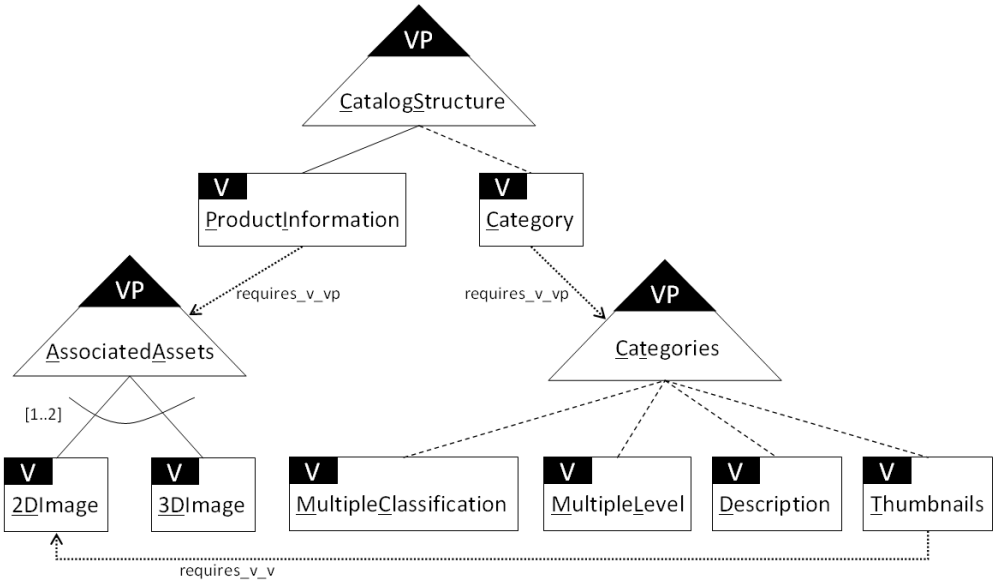


FIGURE 3.2: Exemple du *ItemCatalog SPL* - OVM

3.3 L'approche de modélisation par *features*

Une approche historique d'expression de la variabilité est introduite en 1990 dans la méthode Feature-Oriented Domain Analysis (FODA) par Kang *et al.* [KCH+90] : diagramme de caractéristiques, ou diagramme de *features*. Cette méthode introduit l'utilisation d'un diagramme de caractéristiques (ou *"Feature Diagram"*) à des fins de représentation de la variabilité. FODA trouve ses origines dans plusieurs travaux : ceux de Neighbors sur l'approche Draco [Nei86], l'analyse du domaine de Batory pour les DBMS [BBG+90]. La notion de *features* apparaît dans l'outil ROSE [Lub88, Big89], qui supporte une sélection des caractéristiques ("*feature-based selection*"), et le système KAPTUR [MB89, Bai93], qui met en œuvre "*distinctive features*" afin d'identifier des systèmes d'un même domaine. Il n'existe pas à ce jour de norme ou standard de facto de *Feature Model* et la littérature regorge de définitions variant sur leur sémantique et leur notation (syntaxe diagrammatique principalement) ; quelques exemples [KCH+90, KKL+98, GFA98, CE00, PKGJ08, SHTB07]. La tendance est d'augmenter la capacité d'expressivité des modèles. Des études comparatives ont été établies et Schobbens *et al.* introduisent par leur article [SHTB07] une sémantique générique et spécialisable.

L'apparente simplicité des FMs à mettre en relation des relations entre propriétés, de même que leur intuitivité et efficacité de communication entre différentes parties-prenantes, font des FMs un moyen d'expression de la variabilité populaire dans le domaine des LdPs.

3.3.1 Qu'est ce qu'un *feature* ?

Qu'entendons-nous par *feature* ? De manière informelle et somme toute pragmatique, un *feature* est une caractéristique d'un produit à un niveau d'abstraction et à une granularité donnés.

Suivant les approches, le *feature* navigue entre une définition orientée vers l'espace du problème - l'analyse du domaine provenant historiquement de l'ingénierie des exigences - et une définition orientée solution - les approches de programmation par *features* (FOP) sont proches de l'implémentation et défendent un lien fort avec des éléments de code.

Le tableau 3.2 propose un panel de définitions issues de la littérature des *lignes de produits*. Certaines définitions se focalisent sur le point de vue de l'utilisateur et uniquement sur celui-ci, par ex. [KCH+90, KKL+98]. Les approches moins contraintes incluent autant que possible des informations sur le domaine, y englobant toutes les informations significatives pour chaque utilisateur. Une application de la séparation des préoccupations (ou points de vues) peut y être réalisée et permettre une meilleure organisation et compréhension de la variabilité. Des points communs ressortent de ces définitions : la notion d'identification et de représentation. Lee *et al.* soulignent dans [LK04] la différence entre un concept et une caractéristique, débat non résolu dans la proposition. En outre, il est à noter que des auteurs tels Kang envisageaient les diagrammes de *features* comme des artefacts et non comme des modèles. Dans notre approche dirigée par les modèles, le modèle de *features* est une abstraction de la variabilité d'un autre artefact (modèle, code, exigence, etc.).

Définition 7. Feature *: abstraction de la variabilité d'artefacts modélisés de l'espace du problème ou de la solution, ou de leur organisation. Dans un graphe (ou un arbre), le* feature *peut être racine (sans parent), composite, primitif (feuille) ou abstrait.*

Un *feature* abstrait [TKES11] est un élément structurant du modèle qui, dans une approche orientée modèles, n'est pas relié à un quelconque élément de modèle.

Pour conclure, les *features* sont communément utilisés pour spécifier et distinguer des membres de la famille de produits. Alors que les FMs décrivaient originellement des aspects du logiciel

Tableau 3.2: Définitions de *feature* dans les LdPs.

FODA [KCH+90] (Kang et al.)	"A prominent or distinctive and user-visible aspect, quality, or characteristic of a software system or systems"
FORM [KKL+98] (Kang et al.)	"distinctively identifiable functional abstractions that must be implemented tested, delivered, and maintained"
Generative Programming [CE00]	"Property of a domain concept, which is relevant to some domain stakeholder and is used to discriminate between concept instances"
IEEE [oEE90]	"A software characteristic specified or implied by requirements documentation (for example, functionality, performance, attributes, or design constraints)"
Riebish et al. [Rie03]	"A feature represents an aspect valuable to the customer"
Bosch [Bos00]	"a logical unit of behaviour that is specified by a set of functional and quality requirements"
Chen et al. [CZZM05]	"a product characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements"
Batory [Bat05b]	"an elaboration or augmentation of an entity(s) that introduces a new service, capability or relationship"
Batory et al. [Bat05a]	"an increment in product functionality"
Apel et al. [ALMK08]	"a structure that extends and modifies the structure of a given program in order to satisfy a stakeholder's requirement, to implement and encapsulate a design decision, and to offer a configuration option"

traductions p. 205

visible de l'utilisateur, leur usage a évolué pour décrire toutes les commonalités et variabilités d'un artefact à toute partie-prenante.

3.3.2 Les modèles de *features*

Depuis la proposition originelle dans la méthode FODA [KCH+90], de nombreuses extensions ont vu le jour, se différenciant tant sur la syntaxe abstraite (par ex. opérateurs de décompositions - AND, OR, Xor et $V_{i,j}$ [SHTB07], les cardinalités de groupe [RBSP02], les cardinalités des *features* [CHE05] et les attributs) que sur la syntaxe concrète, allant de représentations graphiques (le plus commun) à textuelles. Le *Feature Diagram* (ou diagramme de caractéristiques) est un élément essentiel du *feature model* et concerne sa représentation graphique. L'édition et la visualisation de FMs d'une grande envergure est un défi, traité par de nombreux outilleurs par des fonctionnalités de filtrage et de développement / réduction ("*expand/collapse*") de nœuds [AC04, Kru07, Beu08, KTS+09]. Des approches plus avancées ont été proposées, par ex. [NTB+08, CHBT10], pour faciliter la navigation et compréhension de FMs de grandes tailles.

Les définitions de modèles de *features* sont pléthores et une étude menée par Schobbens *et al.* [SHT06] offre un aperçu des principales approches de modélisation par *features*. Les extensions visent à augmenter la précision de la notation (ajout de sucre syntaxique) et/ou son expressivité (permettant l'expression de nouveaux concepts mathématiques). Les travaux de l'université de Namur aboutissent à [SHTB07], qui présente une sémantique abstraite générique permettant

l'expression de la définition d'un modèle de *features*.

D'aucuns, depuis Mannion [Man02], ont mis en exergue la possibilité et le besoin de traduire les modèles de *feature* dans des formules propositionnelles [Bat05a], et par là bénéficier de la faculté de raisonnement mathématique de divers *solver* [BSC10]. Une formule propositionnelle consiste en un ensemble de symboles primitifs ou de variables et d'un ensemble de propositions conjonctives contraignant les valeurs des variables : $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$. Ces analyses permettent de vérifier, pour partie, les propriétés suivantes :

- La satisfiabilité du FM : un FM est dit satisfiable quand il exprime au moins une configuration (un produit) dérivable. Un FM non-satisfiable est un modèle sur-contraint à partir duquel aucun produit n'est dérivable.
- La validité d'une configuration : une configuration correspond à une sélection d'un ensemble de *feature* du FM ; cette configuration est dite valide si elle respecte toutes les contraintes exprimées par le FM.
- L'absence de *feature* mort : un *feature* n'apparaissant dans aucune possibilité de configuration est dit mort.
- Le nombre de produits dérivables : établi à partir du calcul du nombre exhaustif de configuration réalisable.

3.3.3 Définition du modèle de *features*

La définition s'appuie sur les travaux de Schobbens *et al.* [SHTB07] et pour l'implémentation, un métamodèle est exprimé en ECore et contraint par des annotations OCL (cf. 13.2).

La formalisation a été réalisée suivant les prescriptions de Harel et Rumpe [HR00], selon lesquels, la définition d'un langage de modélisation L doit posséder trois éléments distincts : un domaine syntaxique $\mathcal{L}_L$, un domaine sémantique $\mathcal{S}_L$ et une fonction sémantique $\mathcal{M}_L : \mathcal{L}_L \to \mathcal{S}_L$, ou généralement symbolisé par $[\![\cdot]\!]_L$.

Tableau 3.3: Opérateurs de décomposition des modèles de features

Opérateur	and $\wedge$	or : $\vee$	xor : $\oplus$	$V_{i,j}$	option
Cardinalité	$\langle n..n \rangle$	$\langle 1..n \rangle$	$\langle 1..1 \rangle$	$\langle i..j \rangle$	$\langle 0..1 \rangle$

Définition 8. *Domaine syntaxique* $\mathcal{L}_{FM}$
$d \in \mathcal{L}_{FM}$ *est un 6-uplet* $\langle F, P, r, DE, \lambda, CE, \Phi \rangle$ *tel que :*

- F *est un ensemble non vide et fini de* Features *;*
- $P \subseteq F$ *est un ensemble de* features *primitifs (c.-à-d. des feuilles) ;*
- $r \in F$ *est le nœud racine du FM ;*
- $DE \subseteq F \times F$ *est l'ensemble fini de relations de décomposition* (Decomposition Edges) *entre les* features *qui forment un arbre. Comme les FMs sont directs, les* features $f_1, f_2 \in F, (f_1, f_2) \in E$ *sont par la suite notés* $f_1 \to f_2$ *où* f_1 *est le parent et* f_2 *l'enfant.*
- $\lambda : F \to \mathbb{N} \times \mathbb{N}$ *indique le type de décomposition d'un* feature*, représenté par une cardinalité* $\langle i..j \rangle$ *ou* i *indique le nombre minimal de sous-*features *requis dans un produit, et indique le nombre maximal. Le tableau 3.3 donne un résumé des opérateurs "classiques" et des cardinalités associées.*
- Φ *est une formule qui capture les contraintes transverses (par ex.* requires *ou* excludes*). De manière générique,* Φ *est considérée comme une conjonction de formules booléennes sur les* features*, c.-à-d.* $\Phi \in \mathbb{B}(F)$.

En outre, chaque $d \in \mathcal{L}_{FM}$ *doit satisfaire les règles de bonne formation suivante :*

- $r \in F$ *est le nœud racine et n'a pas de parent :* $\forall f \in F.(\nexists f' \in F.f' \to f) \iff f = r$ *;*

- *DE est acyclique :* $\nexists f_1, \ldots, f_k \in F.f_1 \rightarrow .. \rightarrow f_k \rightarrow f_1$ *;*
- *DE est un arbre :* $\nexists f_1, f_2, f_3 \in F.f_1 \rightarrow f_2 \wedge f_3 \rightarrow f_2$ *;*

L'ensemble des combinaisons possibles des *features* d'un FM donné est appelé *domaine sémantique*, et est défini comme suit :

Définition 9. *Domaine sémantique* $\mathcal{S}_{FM}$
$\mathcal{S}_{FM} \triangleq \mathcal{P}(\mathcal{P}(P))$*, indique que tout diagramme syntaxiquement correct doit être interprété tel une* ligne de produits*, c.-à-d. comme un ensemble de configurations ou encore comme un produit (ensemble des ensembles des* features *primitifs).*

La fonction liant le domaine syntaxique et les configurations valides correspondantes est réalisée par la fonction sémantique.

Définition 10. *Fonction sémantique* $[\![d]\!]_{FM}$
$\mathcal{M} : \mathcal{L}_{FM} \rightarrow \mathcal{S}_{FM}$
où $\mathcal{M}(d)$ *est l'ensemble des ensembles des* features *valides* $c \in \mathcal{P}(F)$*. En considérant* $children(f)$ $f'|(f, f') \in DE$ *comme l'ensemble des sous-*features *de* f*, chaque* $c \in \mathcal{M}(d)$ *est tel qu'il :*

- *contienne l'élément racine :* $r \in c$
- *satisfasse le type de décomposition :* $f \in c \wedge \lambda(f) = \langle i..j \rangle \Rightarrow i \leq |children(f) \cap c| \leq j$*,*
- *justifie chaque* feature *:* $f' \in c \wedge f' \in children(f) \Rightarrow f \in c$*,*
- *satisfasse les contraintes additionnelles :* $c \models \Phi$*.*

Toujours sur le cas d'illustration *ItemCatalog SPL*, introduit dans la section 3.2.4, la figure 3.3 présente une représentation du style FODA [KSP09]. De manière abstraite, le modèle peut s'exprimer sous la forme :

- $F = \left\{CS, PI, AA, 2D, 3D, \dot{C}, C, \dot{MC}, MC, \dot{ML}, ML, \dot{D}, D, \dot{T}, T\right\}$;
- $P = \{2D, 3D, MC, ML, D, T\}$; $r = CS$;
- $DE = \{(CS, PI), (PI, AA), (AA, 2D), (AA, 3D), (CS, C), (C, MC), (C, ML), (C, D), (C, T)\}$
- $\lambda(CS) = \langle 1..2 \rangle$; $\lambda(PI) = \langle 1..1 \rangle$; $\lambda(AA) = \langle 1..2 \rangle$; $\lambda(2D) = \langle 0..0 \rangle$; $\lambda(3D) = \langle 0..0 \rangle$; $\lambda(\dot{C}) = \langle 0..1 \rangle$; $\lambda(C) = \langle 0..4 \rangle$; $\lambda(\dot{MC}) = \langle 0..1 \rangle$; $\lambda(MC) = \langle 0..0 \rangle$; $\lambda(\dot{ML}) = \langle 0..1 \rangle$; $\lambda(ML) = \langle 0..0 \rangle$; $\lambda(\dot{D}) = \langle 0..1 \rangle$; $\lambda(D) = \langle 0..0 \rangle$; $\lambda(\dot{T}) = \langle 0..1 \rangle$; $\lambda(T) = \langle 0..0 \rangle$;
- $\Phi = (T \Rightarrow 2D)$;

Avec $\dot{F}$ encodant l'optionalité d'un *feature*, soit avec une cardinalité de $\lambda(\dot{F}) = \langle 0..1 \rangle$.

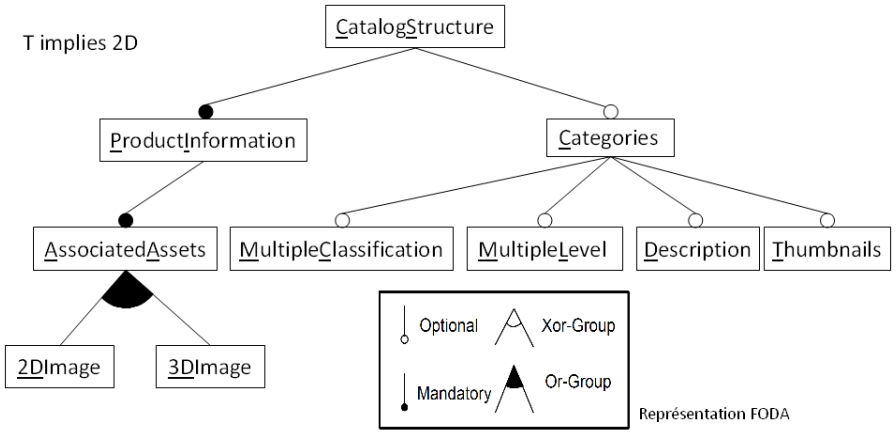


FIGURE 3.3: Exemple du *ItemCatalog SPL* - Représentation de type FODA [KSP09]

3.4 Modularité et relations dans les modèles de variabilité

Les habilités cognitives de tout à chacun sont mises à mal par la complexité des modèles de variabilité, de part leur taille, la richesse d'information et le manque de règles d'organisation dans leur sémantique. Confronté à ce problème, de nombreuses approches mettent en œuvre la tactique de séparation des préoccupations (cf. 2.2). Comme le soulignent de nombreuses études, par ex. [BTN+08, RW07, MPH+07, HTM09, LK10], cette problématique est essentielle au devenir des *lignes de produits*.

Précédemment, la section 3.1.2 proposait une liste de type de variabilité et donc de séparations de préoccupations. La section courante précise cette description et se focalise sur les approches orientées modèles de *features* (section 3.4.1). Par la suite, une description exhaustive des relations explicites dans les approches de modélisation de la variabilité est présentée en section 3.4.2. La description est séquentielle et organisée en catégories d'approches et préoccupations et par ordre de publication. Les approches de modélisation logique de la variabilité, du type formules propositionnelles [Man02], ne font pas l'objet de cette description.

3.4.1 Modularité et séparations des préoccupations dans les modèles de *features*

La modularité observée dans les approches de modèles de *feature* est mise en œuvre suite à deux facteurs initiateurs : (i) l'aspiration à une organisation et structuration du modèle, et (ii) l'inclination à une découpe du modèle pour permettre un passage à l'échelle de technique existante. Ces deux catégories sont représentées ci-après.

Il est à noter sur le sujet, la réalisation de Hubaux [Hub12] d'une enquête systématique sur la séparation des préoccupations dans les FMs. La description est organisée par catégories de préoccupations (orientées : extension de FODA, aspects, exigences, points de vue, visualisation et architecture).

Catégorie de *features* et organisation

Un certain nombre de propositions de classification et catégories de *features* se retrouvent dans la littérature. Les trois propositions suivantes considèrent le *feature* comme utile uniquement à la communication avec l'utilisateur final. Une organisation des *features* est déterminée pour ajouter un contenu sémantique au modèle, autre que la variabilité.

Kang *et al.* [KCH+90, KKL+98, LKL02] proposent quatre catégories de *features* organisées en couches :

1. Capacités ("Capabilities") : se sont les capacités des applications avec une perspective utilisateur externe (services disponibles, opérations et propriétés non fonctionnelles) ; influencés par une taxonomie des IHM, les auteurs raffinent ces capacités en fonctionnel, opérationnel et présentation ;
2. Technologies propres au domaine ("Domain technologies") : elles présentent la manière d'implémenter un service ou une opération (décisions d'architecture) ;
3. Techniques d'implémentation ("Implementation techniques") : se sont les fonctions ou techniques génériques pour implémenter les services, les opérations et fonctions du domaine (décisions de conception) ;
4. Environnement de fonctionnement ("Operating environments") : il représente l'environnement dans lequel les applications sont déployées.

Pour sa part, Bailin [Bai93] fait un distinguo entre des *features* : opérationnels, d'interface, fonctionnels, de performances, de méthodologies de développement, de conception et d'implémentation. Enfin, de leur côté, Riebisch *et al.* [Rie03] suggèrent la distinction suivante :

1. Fonctionnel : pour exprimer la manière dont l'utilisateur interagit avec le produit ;
2. Interface : pour exprimer la conformité à un standard ou sous-système ;
3. Paramètre : pour exprimer des énumérations, des listes de propriétés environnementales ou non-fonctionnelles.

Séparation des modèles

Un problème récurrent à de nombreuses approches de LdPs concerne le passage à une échelle industrielle. Les *lignes de produits* de l'industrie peuvent aisément comporter des milliers de points de variation et de paramètres de configurations (par ex. [STB+04]). La gestion d'une variabilité de cette ampleur est extrêmement complexe et requiert des techniques de modélisation sophistiquées.

Certaines approches tendent à gérer cette complexité en appliquant une séparation des préoccupations en plusieurs modèles de variabilité :

- Thompson *et al.* [TH03a] mettent en avant qu'une décomposition hiérarchique unique des *features* ne permet pas de capturer convenablement la structure d'un domaine. Les auteurs proposent des FMs hiérarchiques sur n dimensions ("n-dimensional FMs"). Les dimensions citées sont : la plate-forme physique ("Physical plat-form"), le comportement requis ("required behaviour"), et les capacités de tolérance aux fautes ("fault tolerance capabilities").
- Dans le contexte de l'automobile, [RW06, RW07] se focalisent sur la gestion de large FMs et des changements qui leurs sont apportés dans le temps. Les auteurs proposent une approche nommée "multi-level feature tree", prônant une décomposition hiérarchique des FMs : des FMs locaux pouvant raffiner et spécialiser une structure de LdP de base sans en modifier le contour. Hartmann *et al.*
- [HT08, HTM09] se confrontent au problème de la gestion de multiple LdPs. Ces auteurs proposent d'exprimer un FM dédié à l'expression du contexte et un FM dédié aux fournisseurs.
- Grünbacher *et al.* discutent des défis de la structuration de l'espace de modélisation des LdPs [GRDL09]. Ils argumentent sur l'infaisabilité de maintenir un unique FM et suggèrent des stratégies de décomposition de la modélisation par *features* suivant plusieurs perspectives.
- Dans leur travaux, Lee et Kang [LK10] introduisent une modélisation des usages du contexte et le relient au modèle de *features* produit afin d'améliorer le processus de dérivation.
- L'approche de Acher [ACLF10, Ach11] met en œuvre de multiple modèles de *features*, orchestrés par une sémantique opérationnelle de composition et de découpe.

D'autres approches, telle celle de Griss *et al.* [GFA98] qui introduit différentes vues sur un FM, centrent les débats sur les techniques de visualisation :

- Nestor *et al.* [NOQ+07, NTB+08] de leur côté, cherchent à résoudre ce problème en se focalisant sur une approche orientée visualisation de l'information. Les techniques de visualisation de l'information assistée par le logiciel apportent un support à la visualisation de données pour amplifier le domaine cognitif.
- [BTN+08, CTH08] proposent une discussion sur l'apport des techniques de visualisation pour l'exploration de FMs complexes. L'outillage associé fournit les services suivants : affichage des détails sur demande, survol incrémental, ciblage d'un élément de contexte et affichage de la configuration d'un produit.

- Gonzalez-Baixauli *et al.* présentent une approche d'analyse visuelle basée sur l'examen des relations entre les objectifs non-fonctionnels et fonctionnels des exigences, ainsi qu'un support outillé [GBLM04].

3.4.2 Différentes relations dans les modèles de variabilité

Cette section présente les résultats d'une étude exposée à JLdP 2012 [CC12], visant à recenser les approches de variabilité présentes dans la littérature qui utilisent des relations de manière explicite. Ces approches sont classées par techniques de variabilité, préoccupation et ordre chronologique de publication. Ce paragraphe étend [CC12], en exposant notamment les travaux autour de Dopler (par ex. [DGR11], *decision modeling*), de la gestion par contraintes (par ex. [MSD+12]), et de la traçabilité.

Les questions de l'étude portent sur : *QE1* Quelles sont les relations observées ? (nommage et description succincte), *QE2* Quelles préoccupations mettent-elles en relation ? (par ex. préoccupation de variabilité, d'autre type de modèle, inter-préoccupations) *QE3* Quel est leur degré de formalisation ? (texte en langage naturel, sous forme de métamodèle, de formalisme mathématique).

Le protocole de la conduite d'étude est rappelé dans les points suivants :

1. Définition du matériel : une recherche web est conduite depuis Google et dans les librairies ACM Digital Library, SpringerLink, IEEE Explore et Science Direct. Le filtrage initial est ajusté par des mots-clé ciblant les concepts suivants : *relations*, *relationships*, *dependencies*, dans les approches de LdP, de modélisation par *features* et modélisation de variabilité (à l'exception des approches de modélisation par décisions), en utilisant l'élément "*wild card* (*)" et des opérateur booléens.
2. Réalisation d'un filtrage manuel, sur les résumés, introductions, conclusions, puis sur un parcours en diagonale du texte dans sa globalité au besoin. À cette étape, les travaux relatifs aux relations dans des espaces technologiques telles les formules propositionnelles ou la programmation par contraintes sont écartés.
3. Analyse du matériel d'étude en regard des trois questions de recherche, observations, discussions et conclusion.

Les approches présentées ci-après exposent de manière explicite des relations dans des modèles de variabilité. La classification est basée sur les notions de métamodèle et de modèle pour distinguer les approches, (i) de *feature*, ou (ii) d'autre type. Au niveau modèle, un raffinement est appliqué sur un critère intra/inter modèle. Enfin un paragraphe exprimant les besoins de traçabilité dans les *lignes de produits* est ajouté aux résultats de [CC12].

La description ci-après propose une liste d'approches ; dans chaque approche ne sont pas forcement décrites toutes les relations, mais seulement celles significatives (par ex. la relation de contrainte *require* n'est pas répétée dans toute les descriptions).

Relations dans les approches de modélisation par *features*

La diversité des définitions des modèles de *features* (cf. 3.3) fait apparaître différentes relations, à savoir :

- Dans le FM originel, de la méthode FODA [KCH+90], sont introduites une relation structurelle de hiérarchisation *consist-of* et deux règles de composition (*requires* et *mutex-with*). Griss *et al.* [GFA98] conservent ces mêmes relations en adaptant la méthode FODA à RSEB.
- Dans la méthodologie FORM [KKL+98], les auteurs proposent plusieurs catégories de *features*, combinés avec des relations (entre les *features* des différents niveaux) : *composed-of*,

generalization/specialization et *implemented-by*. Ces dernières manquent cependant de formalisation. Des opérateurs logiques ainsi que des dépendances (*requires* et *excludes* - qui est équivalent au précédent *mutex-with*) aident à la structuration du FM.

- Czarnecki et Eisenecker [CE00] suggèrent l'utilisation de dépendances "verticales" entre des *features* de "haut" et "bas" niveau, permettant de gérer différents FMs pour divers types de produits. Dans l'approche, les relations entre un *feature* et son sous-*feature* n'ont pas de sémantique prédéfinie ; la relation est entièrement déterminée par la sémantique du sous-*feature*.
- Fey *et al.* [FFB02] introduisent un métamodèle pour les FMs qui centre son attention sur les relations inter-*features*. Trois plans (ou vues) différents sont définis pour améliorer la représentation de la variabilité : (i) le *plan hiérarchique* (les relations *refine* - similaire à "consist-of" - et *provided-by* sont introduits pour définir le graphe), (ii) le *plan des dépendances* (des relations de cohérence comme *require* et *conflict*, associées à la relation *provided-by* sont utilisées ; ainsi que des relations de modification *modify*, par ex. modification de l'existence ou modification de la valeur), et (iii) le *plan des ensembles de* features.
- Riebish [Rie03], au travers de la méthode Alexandria, décrit le raffinement des relations entre *features* comme *est un* ou *partie de* (dans le texte "is-a" et "part-of").
- Streitferdt *et al.* [SRP03] proposent une formalisation avec OCL des relations des modèles de *features* à des fins de vérification de cohérence de la *ligne de produits*. Les auteurs définissent les contraintes sur les *features* : "requires", "excludes", et "hint", ainsi que la relation "uses".
- Riebish *et al.* [Rie04] réalisent une synthèse des discussions d'un groupe de travail et montrent des relations hiérarchiques présentes dans les FMs (*feature - sub-feature*), ainsi que des contraintes (*requires*, *excludes*). Les auteurs suggèrent également une forme "faible" de ces dernières : *recommends* et *discourages*.
- Dans [MZZ06] (et dans les travaux précédents [ZZM04, ZMZ05]), les auteurs présentent un métamodèle pour modéliser des *features* de niveau système et les relations entre ceux-ci. Les auteurs spécifient explicitement des dépendances statiques et dynamiques entre les *features*. Les dépendances statiques sont spécifiées par des relations de contraintes, et les dépendances dynamiques sont capturées par des relations d'interaction.

Des approches s'intéressent plus particulièrement à ces dépendances entre *features* :

- Ferber *et al.* [FHS02] introduisent une méthode d'investigation sur les dépendances et interactions entre *features* et proposent deux vues complémentaires pour représenter le modèle de *feature* : (i) le raffinement hiérarchique de *features* et (ii) les dépendances et interactions entre *features*. Les auteurs identifient cinq types de dépendances de *features* : intentionnel, usage de ressources, induit par l'environnement, usage et exclusion (dans le texte : "*intentional* interaction", "*resource usage* interaction", "*environment induced* interaction", "*usage* dependency", "*exclude* dependency").
- Lee et Kang [LK04] analysent les dépendances entre *features* et fournissent comme résultat une distinction de six types de dépendances ayant une influence significative sur la conception des artefacts de la LdP. Les auteurs organisent la modélisation de la variabilité en termes de relations structurelles (viz. *aggregation* et *generalization*), de dépendances de configuration (viz. *require* et *exclude*) et de dépendances opérationnelles. Ces dernières se décomposent en : dépendances d'usage (fonctionnelles), dépendances de modification (modification comportementale d'un *feature* par un de ses semblables), et quatre dépendances d'activation. L'attention est dirigée sur l'analyse opérationnelle (à l'exécution), ce qui ne fait pas l'objet de la thèse. Les travaux sont poursuivis et appliqués à la programmation orientée aspects [LBT09], pour se consacrer aux dépendances opérationnelles de modification et de configuration.

- Jaring et Bosch [JB04b, JB04a] proposent une taxonomie des dépendances dans la variabilité, exprimée en propositions logiques.
- Vranić [Vra04] expose une série de connecteurs logique pour des FM (par ex. *implication* $\Rightarrow$, et *equivalence* $\Leftrightarrow$), utilisés pour définir les règles de dépendances par défaut. Les relations entre le FM et les autres artefacts de modélisation sont symbolisées par un ensemble de liens.
- Zhu *et al.* [ZLZZ06] (sur la base de [LYZZ06]) présentent une approche d'analyse de l'influence des dépendances entre les exigences du domaine pour les architectures des LdPs. Les auteurs définissent une classification des dépendances de *features* ; les dépendances statiques sont *decomposition*, *generalization*, *required*, et *excluded* ; les dépendances dynamiques *serial*, *collateral*, *synergetic*, et *change* (state, behave, data, code). Ils établissent également des règles sémantiques de liens entre les *features* et l'architecture.
- Ye et Liu [YL05] fournissent une approche basée sur les matrices avec pour objectif de faire face au problème d'extensibilité des modèles de *features*. Trois relations hiérarchiques (*composition*, *generalization / specialization* et *variation point*) et trois relations non hiérarchiques *requires*, *excludes* et *impacts*) sont identifiées.
- Dans [SFR06], les auteurs proposent d'utiliser des dépendances afin de résoudre la variabilité. L'approche définit la notion de *dépendance fondamentale* (imposée par l'espace du problème), et de *dépendance spécifique à la configuration* (relative à l'implémentation).
- Cho *et al.* [CLK08] introduisent une méthode orientée aspects pour gérer les dépendances opérationnelles entre *features*. Les relations exposées sont basées sur des travaux plus anciens [LK04] (présenté ci-avant).
- Une formalisation des dépendances de [KKL+98] et [LK04], ainsi que l'établissement de deux catégories *global* et *local* est réalisée dans [DS09].

Les approches suivantes définissent des relations liées à la problématique de visualisation des modèles de *features* :

- Botterweck *et al.* [BTN+08] introduisent dans leur approche un métamodèle et de l'outillage de recherche associé qui emploient des techniques de visualisation pour permettre de traiter les tâches fondamentales des développements des LdPs. Les auteurs réifient des relations dans leur métamodèle : *Requiert* est une dépendance directe entre des éléments du même type ("Requires_FF" exprime des dépendances entre *features* ; "Requires_DD" entre des décisions ; "Requires_CC" entre des composants) ; *Exclue* est une dépendance non directive, puisqu'elle décrit un ensemble d'éléments qui sont mutuellement exclusifs ; *ImplémentéPar*, qui également possède des sous-types associés aux différents éléments ("ImplementedBy_DF" exprime que la décision est implémentée par un certain *feature*, et "ImplementedBy_FC" qu'un *feature* est implémenté par un composant donné).
- Zhao *et al.* [ZZM08] abordent le problème de la personnalisation collaborative de FMs en proposant une approche multi-vues. Les auteurs se reposent sur un métamodèle de FM qui réifie des relations de raffinement parent-enfant (*decomposition*, *characterization* et *specialization*), ainsi que des contraintes (*require*, *exclude* et des plus complexes).
- Cawley *et al.* discutent [CTBN09] de la représentation des relations qui existent entre les modèles et présentent une approche pour la communication de ces relations basée sur des techniques de visualisation. Un environnement de visualisation 3D est employé pour exprimer le rendu de ces relations et fournir une aide cognitive aux parties-prenantes lors de la dérivation des produits. Les trois dimensions sont : décision, *feature*, et composant. Les décisions sont implémentées par des relations de *features*, et les *features* sont implémentés par des relations de composant ("Feature requires Feature", "Feature excludes Feature", "Feature recommends Feature" (pas davantage de détails fournis), "Feature problematic with Feature" (pas davantage de détails fournis)).

- Choi *et al.* organisent les FMs suivant cinq vues [CLLK09]. Dans la vue *structurelle* sont représentées les relations telles que l'agrégation et la généralisation ; dans la vue *configuration*, les *features* obligatoires, optionnels et alternatifs sont représentés avec les cardinalités entre *features* et les groupes ; dans la vue *liaison* ("*binding*") apparaissent les unités de liaison agrégeant des *features* ; dans la vue *dépendances opérationnelles* figure une représentation des interactions (dépendances dynamiques) entre *features* ; enfin, la vue *traçabilité* montre les relations *implemented-by* existantes entre les *features*.

La finalité de la modélisation de la variabilité pour les LdPs se trouvant dans la dérivation de produit automatisé, de nombreuses approches définissent des "mapping"[4] entre les *features* et les artefacts d'implémentation :

- Sametinger et Riebisch [SR02] fournissent des informations additionnelles aux FMs sur les détails des artefacts de la solution en liant les *features* aux éléments de conception et d'implémentation.
- Dans [Rie03], ces liens sont exprimés comme des connexions de *features* aux autres modèles sous la forme de liens de traçabilité.
- Czarnecki *et al.* [CA05] associent les FMs aux modèles en leur ajoutant des méta-informations *(tags)*.
- Dans [HKW08], les auteurs utilisent la notion de *mapping* entre *feature* et artefacts de conception, sans en préciser la sémantique. Heidenreich *et al.* [HKW08, Hei09, HSS+10] relient les FMs à divers types de modèles tels que des cas d'utilisation, des diagrammes de classes et des statecharts.

Dans des approches liées à la séparation des préoccupations et à la modularité :

- Dans [RW07], Reiser et Weber sont concernés par la maîtrise de modèles de *features* de grande ampleur et complexité, organisés tels des sous-FMs hiérarchiques *(sublines)*, où des *features* sont connectés entre eux par des relations de *referenceFeature*. Les auteurs utilisent des FMs et fournissent des liens directs et personnalisables *(feature links)* pour contraindre l'ensemble des configurations valides (par ex. *"needs", "suggests", "independent of" ou "excludes"*).
- Comme introduit précédemment, Metzger *et al.* [MPH+07] séparent la *variabilité de la LdP* de la *variabilité logicielle*. Dans [MPH+07, HHU08, TTBC+09], les auteurs définissent des dépendances comme des liens inter-modèles génériques appelés *x-links*. Les modèles de variabilité et les liens sont formalisés pour être analysés comme des problèmes de satisfiabilité. Dans le dernier article [TTBC+09], les auteurs réalisent une séparation entre (i) les exigences, (ii) le contexte du monde du problème, et (iii) les spécifications, avec pour objectif de générer des configurations depuis les exigences sélectionnées. Plus récemment, Classen *et al.* [CHS+10] formalisent les relations entre les FMs et des modèles de conception pour des applications de vérification de cohérence des modèles.
- Czarnecki *et al.* proposent dans [CSW08] de distinguer les contraintes imposées par le modèle ("*hard* constraints") de celles sur la sélection des *features* ("*soft* constraints").
- Bošković *et al.* [BMB+11] proposent une approche visant à améliorer la maîtrise de la gestion des FMs et à réduire les efforts de maintenance. Les auteurs se reposent sur la structure des arbres de *features* et introduisent deux sortes de contraintes d'intégrité, nommément *intra-concern* (contraintes d'intégrité entre des *features* d'une même préoccupation, viz. *includes* et *excludes*) et *inter-concern*(contraintes d'intégrité entre des *features* de préoccupations diverses, introduisant la contrainte d'équivalence : *equivalent*.
- Hartmann et Trew [HT08] introduisent des modèles de variabilité du contexte pour apporter assistance à la gestion de multiples *lignes de produits*. Les relations *requires* et *excludes* sont utilisées dans les FMs, et des liens de dépendance *(dependency links)* sont établis entre le

[4] translation, liens sémantiques

modèle de variabilité du contexte et le FM. Ces liens peuvent être annotés d'indication sur la logique sous-jacente *(rationale)* de l'établissement des contraintes imposées par le contexte (par ex. exclusion mutuelle, m.-à-j. des cardinalités).
- Lee et Kang [LK10] définissent des liens entre leur modèle de description de la variabilité du contexte et les autres modèles de variabilité. L'approche met en application des liens n-m pour connecter les différentes préoccupations.

Relations dans les autres approches de modélisation de variabilité

L'environnement COVAMOF [SDNB04, DSB09] modélise la variabilité en termes de points de vue ("Variation Points - VPs") et dépendances, qui représentent les contraintes sur ces VPs. L'approche définit des entités relationnelles appelées *realization relation entities* pour relier des VPs appartenant à différents niveaux d'abstraction. Les entités de dépendances *(Dependency entities)* spécifient des liens sémantiques entre les sous-configurations des VPs et les domaines cibles. L'approche propose une vue dédiée à la représentation des dépendances.

Pohl *et al.* proposent le langage OVM [PBL05] (cf. section 3.2.4), un DSML basé sur les concepts de point de variation et de variant. L'approche fournit une vue transverse sur la variabilité des artefacts de développement. Des contraintes telles *mutex* et *require* sont représentées et des liens de traçabilité nommés *artefact dependency* sont définis pour relier les artefacts de variabilité à ceux de conception.

Dans les approches orientées description de l'architecture, Liu *et al.* [LLT05] délimitent le pourtour des préoccupations dans la conception architecturale par des connecteurs nommés *concern connector* à des fins d'évolution de la LdP. Les auteurs mettent en avant un cadre pour la séparation des préoccupations basé sur la notion d'"hyperspaces" et des connecteurs pour relier ces préoccupations.

Toujours dans le même domaine des architectures, Thiel *et al.* fournissent quatre extensions aux *IEEE Architecture Descriptions* pour les adapter au contexte des LdPs [TH02]. Les extensions sont : (i) L'extension *ligne de produits* pour capturer les relations entre la ligne de produits, le produit, et l'architecture de la LdP ; (ii) L'extension variabilité des *features* reliant la variabilité aux exigences ; (iii) L'extension variabilité de l'architecture, permettant de connecter la variabilité à l'architecture ; (iv) L'extension des éléments de conceptions, réalisant les relations entre les composants et les types de connecteurs et la variabilité de l'architecture.

La technique de variabilité fusionnée définie dans [MPL+09] est limitée dans l'expression des relations. Typiquement, ce type d'approche repose sur les relations définies par le langage cible faisant l'objet de la fusion. À l'exemple de, dans le domaine de la description de l'architect ure, Asikaisen *et al.* [ASM04] qui proposent avec Koalish une approche hybride qui étend le langage de description d'architecture et de composants Koala [vOvdLKM00] avec des mécanismes d'expression de variabilité.

L'approche textuelle TVL [CBH11] englobe l'expressivité des langages de *features* existants. Sur le plan relationnel, le langage définit formellement les relations *equality* et *implication*, de même que des contraintes avec gardes.

D'autre part, l'approche de modélisation des décisions DOPLER [DGR11], se rapproche des approches par *features* par l'intégration de dépendances hiérarchiques et logiques *(hierarchical and logical dependencies)*, ayant des effets de prise de décision de type *implies*, *bind* et *update*. Du côté *core assets*, des dépendances fonctionnelles et structurelles sont également décrites *(functional and structural dependencies : parent, child, inclusion, exclusion, implementation, abstraction, consists of, contributes to, is predecessor of, is successor of)*. Les auteurs précisent que la relation *parent* peut se raffiner en *contained in*, *constributes to*, *is part of*, *is constituent of*.

En outre, Mazo *et al.* [MSD+12] mettent en exergue l'apport de la programmation par contraintes pour les *lignes de produits*. Les auteurs proposent d'exprimer toutes les informations de variabilité (classiquement contenues dans les modèles de *features*) sous forme de contraintes. Ils expriment ainsi des contraintes de configuration et d'interdépendance : de manière symbolique (par ex. *at most*, *at least*, *exactly*), booléenne (par ex. *equivalence*, *implication*, *xor*) et arithmétique. L'approche met en avant des relations de décomposition hiérarchique, de dépendance de configuration (*requires, includes, exclusion*) et d'inter-modélisation (*conjunction of subgraphs*).

Enfin, avec la volonté de faire cohabiter des modèles hétérogènes dans une approche d'écosystème, et plus spécifiquement centré sur le processus de configuration, Dhungana *et al.* [DSB+11] définissent des dépendances inter-modèles de *features* ("provides feature", "extends feature") et inter-modèles génériques ("alternative models", "extends model").

Relations dans les approches de traçabilité des LdPs

Une discipline concernée par les relations entre divers artefacts est la gestion de la traçabilité. Initialement centrée sur les dépendances entre les exigences et leurs allocations dans des artefacts de conception, la traçabilité couvre un plus large spectre dans les LdPs et plus particulièrement celles dirigées par les modèles. Cette section étend la 2.5 aux approches de *ligne de produits*. Les approches visent principalement à établir des liens de traçabilité entre le modèle de variabilité et les modèles de conception [BW02, JZ05, KCL05, JZ09, CdCMdM+11], tels que les spécifications (cas d'utilisation) ou les modèles d'architecture [MCNY07]. La particularité des applications amène le besoin vers une traçabilité de la variabilité et de ses impacts sur les autres artefacts de la LdP [Moh03] . La LdP est, à l'instar des autres développements logiciels, concernée par l'évolution, et des approches se basent sur la traçabilité pour l'affronter [RP01, AK08, HRDG09].

Une classification de la traçabilité a récemment été publiée [AKM+10]. Anquetil *et al.* définissent une implémentation de mécanismes de variabilité dans un cadre générique et définissent quatre dimensions de variabilité :

- *raffinement* : relations d'artefacts à différents niveaux d'abstraction ;
- *similarité* : relations d'artefacts à un même niveau d'abstraction mais avec une granularité de détails distincte ;
- *temps* : relations liées à la gestion de version ;
- *variabilité* : relations utiles à la gestion de la variabilité.

Ces travaux étendent ceux de Pohl *et al.* [PBL05] (qui classent les traces suivant les ingénieries du domaine et de l'application) et de Berg *et al.* [BBM05] (qui proposent trois dimensions : abstraction, raffinement problème/solution et variabilité).

Les préoccupations de Lamb *et al.* [LJZ11] vont vers la formalisation de la traçabilité et de la sémantique de ces dernières dans le cadre des *lignes de produits* (les types de relations présentés dans la section 2.5.1 sont repris ainsi que des relations de *similarité* et *variabilité*.

Synthèse

L'étude couvre 44 travaux publiés dans des "workshops", conférences, journaux, ouvrages et rapports techniques. Cette répartition est détaillée dans le tableau 3.4 et illustrée par la figure 3.4 B). L'étude propose des résultats qui couvrent une période allant de 1990 à 2012, avec 31% des travaux publiés en 2008-2010, correspondant à une période ou de nombreuses études ont été réalisées sur la cohérence des *FMs* (avec des raisonneurs de type SAT, BDD) ; la figure 3.4 A) illustre cette distribution. Les données brutes et un recueil de tableaux de description des répartitions figurent en annexe B (section B.2).

Sur la base de nos observations de l'étude en concordance avec la littérature sur la traçabilité, les relations des LdPs, peuvent être classées par 5 critères principaux :

Tableau 3.4: Distribution des travaux de l'étude par catégorie de publication.

Type de publication	*Nb de travaux*
Ouvrage	2
Rapport technique	1
Journal	11
Conférence (GPCE, ECMR, SPLC, ECBS, ICSR, SERP, VaMoS, RE, MoDELS, Net.ObjectDays, ICCET, SDL)	23 (SPLC - 12)
Worskhops (SPLC, ICSE, PFE, ECOOP)	7

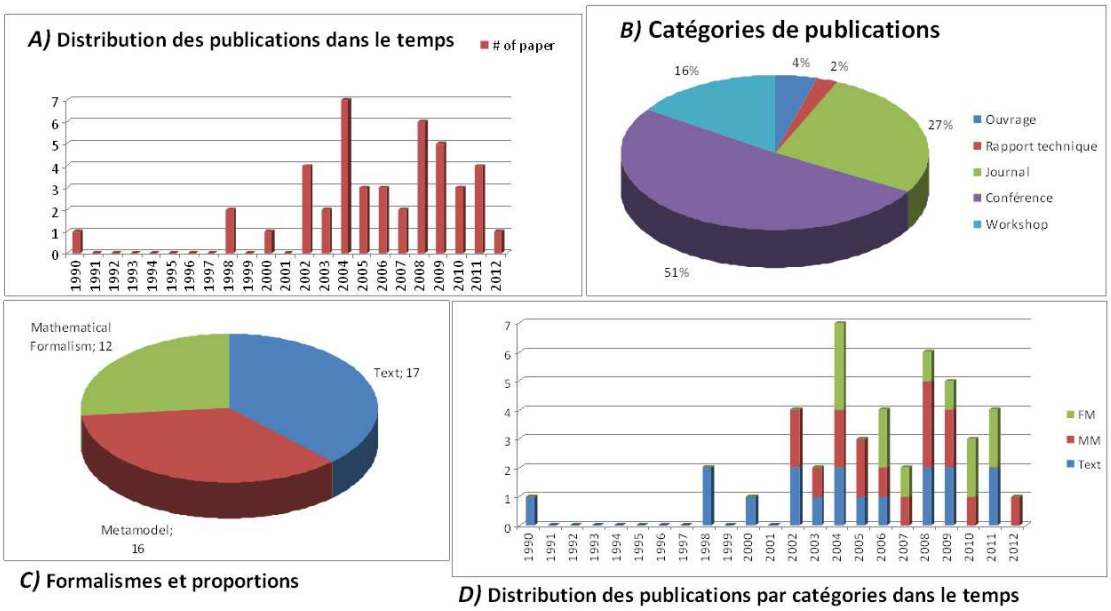


Figure 3.4: Distribution des travaux de l'étude suivant des critères de temps, catégories de publication et de relations

1. *Préoccupation* : déterminant si la relation lie des préoccupations différentes ou non (*intra or inter concern*) ;
2. *Artefacts reliés* : mise en correspondance entre des artefacts de même nature *(homogeneous)* ou non *(heterogeneous)* ;
3. *Niveau d'abstraction* : la relation est définie comme verticale si elle traverse un niveau d'abstraction, ou horizontale dans le cas inverse ;
4. *Temps* : la relation est temporellement reliée à (i) la conception *(design time)*, de type structurelle (provenant de groupes logiques ou d'une hiérarchie) ou dépendance (configuration de base ou spécifique), (ii) opérationnel *(run-time)*, (iii) dans le temps *(over-time)*, relatif à l'évolution.
5. *Impact* : la relation définit (i) une induction (implication, exclusion ou activation), (ii) une modification (interaction, substitution, ou suppression).

Les 5 critères, illustrés par la représentation partielle d'un modèle de *features* dans la figure 3.6, sont combinés pour aboutir à 5 types principaux de relations :

- *Relations de groupes logiques* (*intra-concern, homogeneous, horizontal, structural design,*

co-implication) : elles sont présentes dans la totalité des approches de modélisation de variabilité ;

- *Relations de raffinement hierarchique* (*intra-concern, homogeneous, vertical, structural design, co-implication*) : ce raffinement est typique des modèles de *features*. Initialement nommées *Consist-of* [KCH+90], ces relations sont symétriques et transitives, et sont spécialisées en *generalization* (is specialized by), *aggregation* (has a), *decomposition* (part of) et *classification* (is a). Les approches sont : [KCH+90, GFA98, KKL+98, CE00, FFB02, FHS02, Rie03, LK04, JB04b, JB04a, Rie04, Vra04, CA05, YL05, MZZ06, ZLZZ06, RW07, CLK08, ZZM08, DS09, CLLK09, CBH11, HKW08, MPH+07, CHS+10, CSW08, LK10, DGR11, MSD+12].
- *Dépendances de configuration standards* (*intra-concern, homogeneous, horizontal, design dependencies, implication or inclusion impact,*) : se basent principalement sur les deux contraintes suivantes, *Implication* (dissymétrique et transitive), et *Exclusion* (symétrique mais non transitive). Les approches sont : [KCH+90, GFA98, KKL+98, CE00, FFB02, FHS02, TH02, Rie03, SRP03, JB04b, JB04a, LK04, SDNB04, Vra04, Rie04, ASM04, PBL05, YL05, CA05, MZZ06, SFR06, ZLZZ06, RW07, MPH+07, BTN+08, CSW08, CLK08, HT08, HKW08, HTM09, ZZM08, CTBN09, CLLK09, DS09, MPL+09, DSB09, BMB+11, CHS+10, LK10, CBH11, DGR11, LJZ11, DSB+11, MSD+12].
- *Relations pour des usages spécifiques* (*intra-concern, homogeneous/heterogeneous, horizontal, dependencies, various impacts*) : dépendant d'un usage, les relations peuvent être raffinées par des propriétés sémantiques (bien que très souvent peu précises), tel que pour la relation *Implication* : *depends-on, requires, resource, includes, impacts, satisfies, uses, allocate-to, induces, design, core, implemented-by* ; et pour *Exclusion* : *incompatible, mutex-with, conflict*. Les approches observées sont : [KKL+98, FFB02, FHS02, Rie03, SRP03, LK04, Rie04, MZZ06, SFR06, ZLZZ06, RW07, BTN+08, CLK08, CSW08, ZZM08, CTBN09, CLLK09, DS09, LK10, CBH11, LJZ11, MSD+12]. Ces relations correspondent à divers points de vue (toujours pas forcément bien établis), pour définir par exemple des dépendances opérationnelles, comportementales ou de *(runtime)* [SRP03, LK04, SFR06, ZLZZ06, CLK08, DS09].
- *Dépendances inter-préoccupations* (*inter-concern, homogeneous/heterogeneous, horizontal, dependencies, various impacts*) : dans un modèle de variabilité unique (par ex. [KKL+98, CE00, FHS02, DS09, DGR11, MSD+12]), entre modèles de variabilité (par ex. [SDNB04, PBL05, MPH+07, RW07, HT08, HTM09, DSB09, TTBC+09, BMB+11, LK10, DSB+11, LJZ11]), ou entre modèles de variabilité et autres artefacts de modélisation (par ex. [SR02, SDNB04, PBL05, ZLZZ06, BTN+08, HKW08, CTBN09, TTBC+09]).

La figure 3.5 F) illustre la répartition des catégories de relations dans les travaux sélectionnés, et le grahique E) raffine la distribution des publications dans le temps en y faisant apparaître les catégories de relations et leur évolution.

Il est à noter que jusqu'à 2004, les travaux se sont plus particulièrement intéressés aux dépendances de configuration, pour se concentrer sur l'explicitation des préoccupations et des relations entre elles. Des efforts de formalisation grandissant ont été fournis afin de renforcer la cohérence de la LdP.

Le degré de formalisation est divisé en trois types :

1. *Description textuelle :* [KCH+90, GFA98, KKL+98, CE00, Rie03, Rie04, FHS02, LK04, SFR06, CLK08, CTBN09, CLLK09, SR02, CA05, BMB+11, HT08, HTM09, DSB+11] ;

2. *Représentation par métamodèle :* [FFB02, SRP03, MZZ06, YL05, BTN+08, ZZM08, HKW08, RW07, SDNB04, DSB05, PBL05, TH02, MPL+09, ASM04, DGR11, MSD+12, FHMP+12] ;

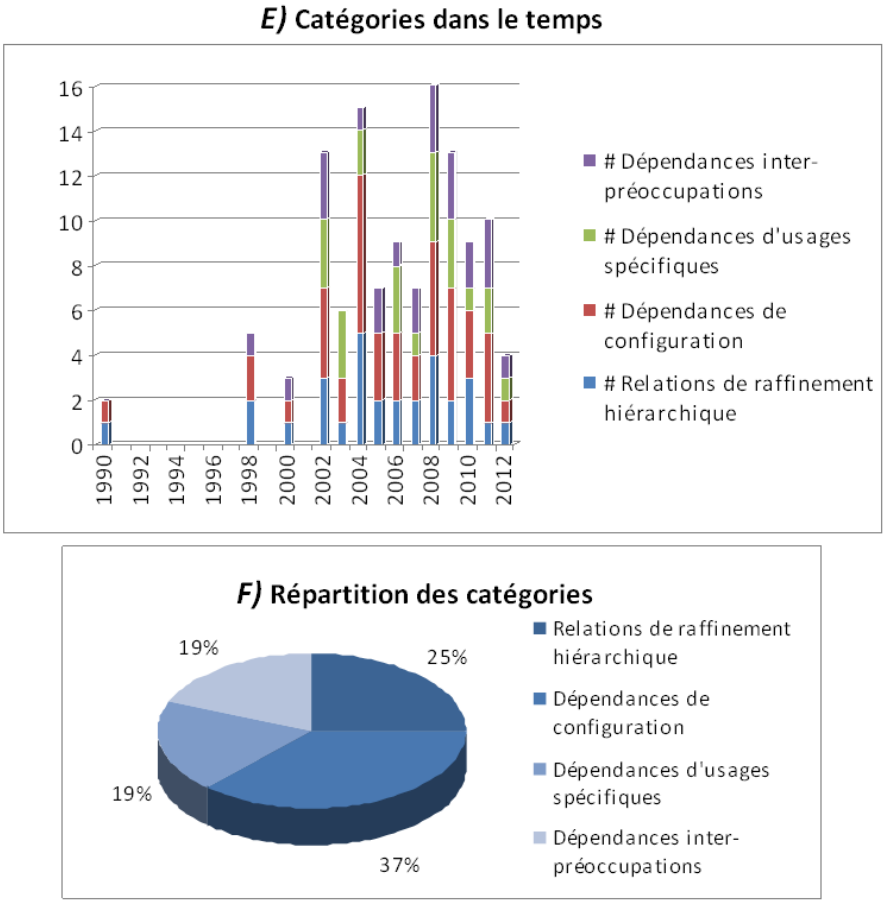


Figure 3.5: Distribution des travaux de l'étude suivant des critères de temps, et de relations

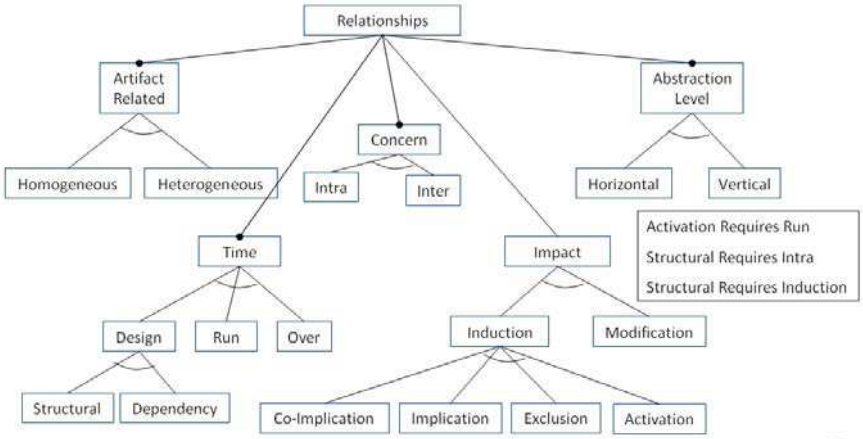


Figure 3.6: Représentation partielle du FM modélisant les 5 critères de classification des relations

3. *Une formalisation mathématique :* qui peut être reliée à des formules propositionnelles ou à un quelconque type de formalisme ensembliste [Vra04, MZZ06, JB04b, JB04a, ZLZZ06, MPH+07, DS09, CHS+10, CSW08, TTBC+09, LK10, CBH11, LJZ11].

La figure 3.4 C) illustre la répartition proportionnelle des formalismes (*Text* pour texte naturel, *Metamodel - MM* pour métamodèle, et *Mathematical Formalism - MF* pour formalisme mathématique) dans les travaux sélectionnés, et le grahique D) représente la distribution des publications dans le temps en y faisant apparaître les formalismes. La répartition globale est équilibrée, avec un plus grand souci de formalisation dans les dernières années.

À titre de conclusion il est intéressant de remarquer que, dans les approches de modélisation de la variabilité, davantage d'efforts ont été apportés dans les approches de *FMs* pour réaliser des séparations des préoccupations. L'objectif premier des modèles de *features* qui se restreint à exprimer uniquement la variabilité a montré ses limites, et de nombreux travaux expriment le besoin d'améliorer l'expressivité des modèles. Néanmoins, il n'y a pour le moment pas de consensus marqué dans la littérature sur les relations dans les modèles de variabilité, et même si des catégories se dessinent parfois, les relations restent cantonnées à être des éléments de second plan. Ce besoin d'exprimer clairement des relations est cependant relevé par certains auteurs, à l'exemple de Botterweck et Lee [BL09].

Le besoin de mettre en relation des variants ou partie de modèles de variabilité entre eux se retrouve dans de nombreux travaux (et plus spécialement dans les approches de modèles de *features*). Malheureusement, ces relations sont souvent enfouies dans les modèles de variabilité, et peuvent présenter un manque de précision dans la sémantique qui leur est associée : leur description peut être relativement vague et imprécise, notamment concernant la description de leurs usages. Des efforts sont donc à fournir pour (i) clarifier les relations, (ii) leur sémantique, et (iii) les usages correspondants.

3.5 Discussion et synthèse

La variabilité est exprimée par divers types de modèles. Les deux modèles les plus couramment utilisés, le modèle de *feature* et OVM [PBL05], ont été présentés en détails dans ce chapitre en réponse à QR 2.

Dans la pratique, la complexité inhérente aux *lignes de produits* amène à séparer les préoccupations, tant dans la modélisation de la variabilité, que dans la modélisation des artefacts de conception. Dans ce contexte, l'information relationnelle apparaît partout, provenant de recouvrements partiels de modèles, d'interactions entre des modèles.

L'étude bibliographique réalisée dans ce chapitre permet de s'apercevoir que l'explicitation de ces relations se retrouve principalement dans la littérature des modèles de *features* (dans une moindre mesure dans les autres approches de variabilité) et dans l'étude de la traçabilité. Cette information relationnelle sépare des préoccupations diverses, avec un degré de précision sémantique variable. Concernant les approches de traçabilité dédiées au cadre des *lignes de produits*, celles-ci s'intéressent peu à la définition de la sémantique et des usages.

Par delà la description des moyens existants de mise en relation dans la représentation de la variabilité, ce chapitre fait émerger le besoin d'explicitation de relations et de modélisation claire des différents espaces de préoccupations.

Les manques suivants sont identifiés :

B 2. *La modélisation de la variabilité présente un manque de conceptualisation claire des artefacts de variabilité permettant de soutenir une définition possible des relations entre ces artefacts et les autres concepts de modélisation. Celui-ci se doit d'être corrigé.*

B 3. *La modélisation de la variabilité affiche un manque d'approche globale de modélisation relationnelle de la variabilité (ergo de la commonalité) dans le contexte de l'IDM, avec une formalisation et une sémantique des relations clairement établies, qu'il convient de pallier.*

CHAPITRE 4

Lignes de produits et évolution

"Obsolete your own products. Grow or die."
—Joseph Irwin Miller

DAns le cadre du développement de systèmes embarqués critiques, les architectures logicielles doivent être construites, de manière à soutenir des fonctionnalités de plus en plus complexes. Les développements sont notamment sujets à des contraintes toujours plus fortes de qualité, de fiabilité et de temps de réalisation.

Un moyen de faire face à la complexité du développement est d'élever le niveau d'abstraction des concepts manipulés, ce qui est le credo de L'Ingénierie Dirigée par les Modèles (cf. chapitre 2). Une autre perspective est de capitaliser sur le savoir-faire métier et de partager des processus d'une part et des artefacts produits de l'autre. La philosophie générale des Lignes de Produits Logiciels (LdPs) est la réutilisation intra-organisationnelle par l'exploitation explicite et planifiée de ressemblances parmi des produits liés. Une LdP est une famille de produits logiciels qui sont construits de manière prescriptive à partir d'un ensemble commun d'artefacts de développement [PBL05, CN01]. La gestion de la modélisation de cette commonalité et variabilité (cf. 3) est centrale aux LdPs.

D'aucuns, à l'instar de Lehman [Leh80], s'accordent, sur la base d'observations empiriques, sur la définition de règles concernant les changements des logiciels au cours du temps. L'évolution est indispensable ; la première loi de Lehman est la *Law of Continuing Change*, comprenez par là que le logiciel est assujetti à une évolution permanente pour ne pas devenir obsolète.

La *ligne de produits* n'échappe pas à cette règle. La particularité de cette approche est associée à la gestion de la variabilité, qui implique de la maintenance [Swa76, Pig97] de variants et un processus de peuplement et régulation de la *ligne de produits* (évolution du périmètre). Confrontées à l'évolution, les *lignes de produits* nécessitent des principes et règles architecturales forts pour garantir la compositionalité des produits [Bos00].

Le chapitre présente le processus des *lignes de produits* en section 4.1 , pour y détailler l'ingénierie du domaine et les particularités de l'architecture d'une LdP en section 4.2. L'ingénierie complémentaire de l'application et les spécificités de la réalisation d'un système logiciel à partir d'une LdP sont présentées section 4.3. Enfin, l'évolution des LdPs, cas particulier de l'évolution des systèmes, est présentée dans la section 4.4.

Ce chapitre répond également partiellement à la question de recherche 2 : *Quels sont les artefacts manipulés ?* Mais également à une question relative à notre motivation principale :

QR 3. *Quelles sont les particularités d'une évolution d'une* ligne de produits *?*

4.1 Le processus des *lignes de produits*

Pour aboutir à un processus de réutilisation systématique, la LdP repose sur deux phases clés, à savoir l'*ingénierie du domaine (Domain Engineering)* et l'*ingénierie de l'application (Application Engineering)* [WL99, PBL05]. Dans la communauté des exigences, Faulk [Fau01] marque clairement la séparation entre ces deux étapes. La première phase fournit un ensemble organisé de documents réutilisés lors de la seconde pour la réalisation d'un produit.

La gestion de la modélisation de cette commonalité et de cette variabilité est centrale aux LdPs ; cette activité fait partie de la phase d'*ingénierie du domaine*, la seconde phase étant l'*ingénierie de l'application*, cf. figure 4.1, qui elle se rapporte à la réalisation, ou dérivation, par composition, adaptation et spécialisation d'un produit particulier à partir des données de la LdP. Des efforts sont portés dans l'*ingénierie du domaine*, afin d'assurer une cohérence et une validité des artefacts et de pouvoir assurer une dérivation des produits de manière optimale.

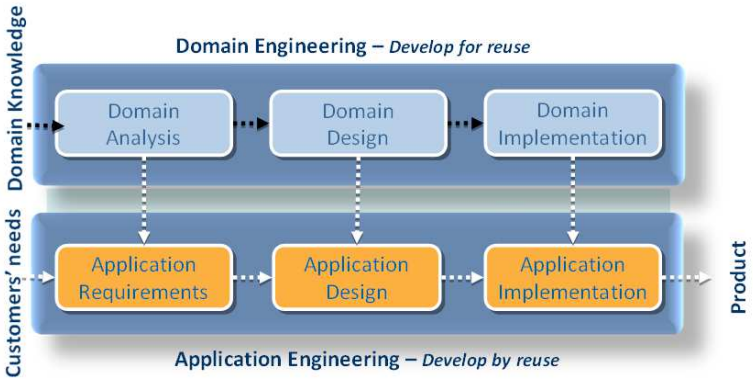


FIGURE 4.1: Processus de *ligne de produits*.

En comparaison des développements traditionnels, l'approche par Lignes de Produits promet de nombreux bénéfices [CN01, PBL05]. De nombreuses expériences de réussites industrielles sont rapportées dans la littérature dans des domaines variés [CN01, BCK03, Beu03, PBL05, KSP09, 9].

Les *Lignes de Produits Dirigées par les Modèles (LdPDM)* (par ex. [CAK+05, Jør06, PKGJ08, MVL+08, TBD07, VG07] combinent les bénéfices de l'IDM et des LdPs. S'ajoutent à cela les bénéfices comme une vérification de la plate-forme de construction et l'optimisation du temps de production [TBD07, BAS08].

4.1.1 Définitions

La notion de *ligne de produits* dans sa forme actuelle provient de la communauté du logiciel. Par conséquent, et bien que l'approche s'étende maintenant aux approches systèmes plus globales, les définitions font références à des LdPs logicielles.

Ligne de Produits

Clements et Northrop [CN01] définissent le concept de LdP logicielle comme un ensemble de systèmes à logiciel prépondérant partageant de manière cohérente un ensemble de caractéristiques communes. Ces dernières satisfont aux besoins spécifiques d'un segment particulier du marché ou à une mission, et sont développées depuis un ensemble commun d'actifs clés de manière prescriptive.

> *"A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfiy the specific need of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way."*
>
> Clements et Northrop [CN01] (traduction p. 206)

Jan Bosh définit ce même concept de manière quelque peu différente [Bos00] : Une ligne de produit consiste en l'agrégation d'une architecture de LdP et d'un ensemble de composants réutilisables, conçus pour être incorporés dans l'architecture de la LdP. La LdP étant, en somme, définie par l'ensemble des produits logiciels développés depuis l'architecture et les composants.

> *"A software product line consists of a product line architecture and a set of reusable components that are designed for incorporation into the product line architecture. In additon, the product line consists of the software products that are developed using the mentioned reusable assets."*
>
> Jan Bosch [Bos00] (traduction p. 206)

Ces définitions partagent la notion d'"éléments réutilisables" permettant la réalisation de "produits systèmes". La gestion prescriptive de la LdP est indispensable à la réalisation d'éléments réutilisables incorporant une architecture prédéfinie, les deux définitions mettant l'accent sur le côté anticipation et réflexions amont de l'approche. La définition de Clements et Northrop suit un point de vue orienté marché et "marketing" alors que la vision de Bosch et ses intérêts l'oriente vers des aspects plus technologiques de réalisation et d'architecture. Les *lignes de produits* sont une conjonction des préoccupations de réutilisation (suite des approches composants [HC01, Szy02]) et de gestion de production de masse ("mass customization" [PD99]).

Définition 11. *Ligne de Produits : Une* ligne de produits *propose une mise en œuvre de moyens techniques visant à satisfaire une réutilisation intra-organisationnelle planifiée, dont le périmètre est délimité par des objectifs de missions ou de marché. Elle permet l'exploitation de similarités entre produits centrés autour d'un même cadre architectural.*

Lors de l'émergence de cette philosophie de réutilisation, les acteurs américains et européens ne travaillaient pas de concert, jusqu'en novembre 1996 (réunion à Las Navas, Spain [vdL02]), il persiste dans la littérature différentes terminologies. Une unification du vocable s'est réalisée au cours du temps et dans les approches contemporaines, la notion de famille fait référence à l'ensemble des produits dérivables par la *ligne de produits*, qui englobe la notion de processus et de moyens de réalisation.

Core Assets et produits

La terminologie anglaise de *Core assets* (introduite dans les définitions présentées ci-avant [CN01, Bos00]) est conservée pour faire référence aux artefacts de conception évoqués dans le chapitre 3. Dans une approche IDM, ces *Core assets* sont des modèles à diverses étapes du processus de développement. Cela inclut, sans se limiter à, des éléments d'architecture, des composants logiciels réutilisables, des modèles de domaines, des exigences, de la documentation et des spécifications, des modèles de performances, des budgets, des plans de tests, et des descriptions des processus.

Les développements mono-application sont dans ce manuscrit opposés aux développements de type *lignes de produits*. L'application résultante du premier processus est qualifiée de monolithique en opposition aux produits résultants d'un assemblage plus flexible amené par la LdP.

Un système logiciel monolithique est donc un tout dont les collaborations entre les différents composants est rigide et difficile à maintenir, voir à étendre. En dépit de ce que son nom pourrait amener à penser, il ne s'agit pas d'un système réalisé par un seul bloc (soit un composant unique), car tout système, par définition[1], est constitué de multiples composants. Ce qualificatif de monolithique souligne la complexité dans l'entrelacement des fonctionnalités et constituants, et insiste sur le fait qu'il n'est pas possible d'identifier clairement des composants à proprement parler, et que par voie de conséquence, il est impossible de faire ressortir des patrons de conceptions.

Définition 12. *Produit : système résultant de l'*Ingénierie de l'Application*, incluant éventuellement des développements spécifiques "produit" permettant une spécialisation non couverte par la LdP initiale.*

Soit T la fonction de transformation de dérivation de produit, et β une fonction d'association des variants V_i (des modèles de variabilité) aux Core Assets. *Soit $\Delta_{specialization}$ la part des développements spécifiques "produit". En considérant une configuration c, comme un ensemble valide de n variants sélectionnés ($c = V_1, \ldots, V_n$), le produit p est défini tel que :*

$$p = T\left(\bigcup_{i=1}^{n} \beta(V_i)\right) + \Delta_{specialization} \cdot$$

En corollaire, ayant défini un produit, la famille de Produits peut être définie par extension.

Définition 13. *Famille de Produits : ensemble des produits réalisés à partir d'une transformation T. L'unicité de la fonction de Transformation préserve l'identité de la LdP et par conséquence de la famille.*

$$Famille_p \triangleq \sum p_j \ , avec(j \in D,\ ensemble\ des\ produits\ dérivés).$$

4.1.2 Un processus d'ingénierie duale

La figure 4.1 présente les deux ingénieries, et pour chacune d'elles trois activités. La notion de domaine recouvre un secteur de métier ou de technologies ou des connaissances caractérisées par un ensemble de concepts et de terminologies compréhensibles par les utilisateurs de ce secteur [Elk01]. La notion d'application est une référence directe à l'application produit.

Analyse du domaine

L'idée d'une ingénierie du domaine et d'une analyse du domaine en vue d'une réutilisation dans des familles de logiciels a été introduite au milieu des années 1970. Le terme "domain analysis" a

[1] l'éthymologie du nom système, du grec σύστημα, le définit comme un tout composé de plusieurs parties membres

été explicité par Neighbors [Nei86] (méthode Draco) et défini telle l'activité d'identification des objets et opérations d'une classe de systèmes similaires appartenant à un domaine de problème particulier[2]. Dès lors, de nombreuses méthodes ont vu le jour : Feature Oriented Domain Analysis (FODA) [KCH+90], [PD91], Feature Oriented Reuse Method (FORM) [KKL+98], Domain Analysis and Reuse Envirronment (DARE) [FPDF98], [CHW98], FeatuRSEB [GFA98], Family-oriented Abstraction Specification and Translation (FAST) [WL99], ProdUct Line Software Engineering (PuLSE) [BFK+99], KobrA [ABB+02], SEI Product Line Practice Initiative (PLPI) [CN01, 9], Feature Oriented Product Line Software Engineering (FOPLE) [KLD02], Quality-driven Architecture Design and quality Analysis (QADA) [Mat04], Evolutionary Software Product Line Engineering Process (ESPLEP) [Gom04], Product Line Use case modeling for Systems and Software engineering (PLUSS) [EBB05], Organization Domain Modeling (ODM) [SCK+96].

Concernant la modélisation de la variabilité, de nombreuses techniques ont vu le jour et ont été présentées dans le chapitre 3.

Conception et Implémentation du domaine

La conception du domaine fait référence à l'architecture de la ligne de produit, la section 4.2 lui est dédiée, et l'implémentation correspond à la prise en considération des préoccupations de plate-forme et à la réalisation de composants réutilisables [HC01, Szy02], voire disponibles sur étagère (*components off the shelf - COTS*) [Cle01]. La problématique de validation et vérification est présente comme dans tout autre développement logiciel, par ex. [GLS08].

Certaines approches listées ci-avant proposent également des solutions de conception du domaine, par exemple [KKL+98, WL99, ABB+02, Gom04, Mat04]. Concernant l'architecture de la *ligne de produits*, des méthodologies sont centrées architecture [Bos00], des langages de description d'architecture (*Architecture Description Languages (ADL)*) tel KOALA [vOvdLKM00] ont été adaptés aux *lignes de produits* [ASM04] (Koalish).

Ingénierie de l'application

Il s'agit d'un développement par la réutilisation. Comme indiqué sur la figure 4.1, les exigences issues du besoin utilisateur sont étudiées en regard des résultats de l'analyse du domaine, une architecture du produit est ensuite dérivée et l'application implémentée. Les points de variations sont résolus étape par étape [SG10], éventuellement par plusieurs responsables [CHE04]. La section 4.3 propose un aperçu des approches de dérivation de produits.

4.2 Ingénierie du domaine et architecture

L'ingénierie du domaine est concernée par la conception et l'implémentation du domaine et, par transitivité, par l'architecture. La qualité du logiciel est aussi associée à la capacité à maîtriser la complexité du logiciel en vue de produire un logiciel robuste, évolutif, maintenable, réutilisable et interopérable ; tel est le sacerdoce de l'architecture.

Architecture est un terme relativement populaire dans la communauté logicielle et son usage peut varier en fonction du contexte. Il désigne des composants logiciels, la structure de l'unité centrale, la structure organisationnelle d'un système d'information. Il existe également plusieurs interprétations et définitions de ce terme dans le contexte des composants logiciels.

[2]"the activity of identifying the objects and operations of a class of similar systems in a particular problem domain"

4.2.1 Définition

De nombreuses définitions de la notion d'architecture parsèment la littérature[3] et il n'est pas aisé d'en synthétiser une unifiante.

L'annexe B (section B.3.1) propose une étude plus détaillée sur la notion d'architecture, mais en substance, l'architecture logicielle peut être comprise d'au moins deux manières différentes : (i) comme un ensemble de décisions (architecturales) de conception, ou (ii) comme la structure résultante de ces décisions. Dans cette thèse, la notion d'architecture s'apparente à la seconde définition. L'essence abstraite de l'architecture logicielle se distingue de la réalisation matérielle représentée dans le programme final.

4.2.2 Conception de l'architecture

La conception de l'architecture est une activité antérieure à toute implémentation et son objectif est d'assurer la structuration de l'ensemble du système logiciel. Dans la pratique sont fréquemment observées des conceptions d'architecture qualifiées de *ad hoc*, idiosyncratiques ou informelles. Il en résulte une architecture parfois peu comprise des développeurs, certains choix architecturaux ne pouvant pas être analysés en cohérence et en complétude. Enfin, autre problème majeur, les choix de conceptions initiaux se révèlent rapidement illisibles quand le système évolue.

Comme le présentent Shaw et Garlan [SG96], de nombreux patrons et styles architecturaux (introduits par Perry *et al.* [PW92]) peuvent être employés à cette étape de conception. Pour plus de détails sur la définition et pour des exemples de patrons d'architecture, se référer à l'annexe B. Notons par ailleurs que, poussé à son paroxysme comme dans l'approche de Ackerman et Gonzales [AG10], le patron se mue en un paradigme de programmation.

4.2.3 Architecture de la LdP

Contrairement aux applications monolithiques, l'architecture concerne ici l'ensemble de la *ligne de produits*. L'architecture constitue l'empreinte de la *ligne de produits* et définit la manière dont les composants logiciels y sont assemblés. L'architecture se doit de supporter la variabilité inhérente à la *ligne de produits*. Pohl *et al.* [PBL05] définissent la PLA *("Product Line Architecture - PLA")* comme :

> *"[...] a core architecture that captures the high level design for the products of the SPL, including the variation points and variants documented in the variability model."*
>
> Pohl *et al.* [PBL05] (traduction p. 206)

La définition du SEI est la suivante :

> *"The product line architecture is an early and prominent member in the collection of core assets. (...) The architecture defines the set of software components (...) that populates the core asset base. The product line architecture - together with the production plan - provides the prescription (...) for how products are built from core assets."*
>
> SEI [9] (traduction p. 206)

[3] Le Software Engineering Institute - SEI - (http ://www.sei.cmu.edu/architecture/published_definitions.html) en dénombre vingt-sept.

Cette définition reprend les aspects structurel et décisionnel des définitions de l'architecture logicielle.

Bass *et al.* [BCK03] décrivent l'architecture de la Ligne de Produits comme une architecture unique réutilisée pour un ensemble de produits. Il s'agit donc de construire une plate-forme utilisable par toute la Ligne de Produits. En effet, selon Bosch [Bos02], les *lignes de produits* logicielles sont typiquement (pour le moins) basées sur des plate-formes, définies par un ensemble de composants logiciels commun à tous les membres de la LdP.

D'autre part, de récentes approches [PBL05, NAB11a, NAB11b] explorent l'application d'architectures de référence aux *lignes de produits*.

De nombreux auteurs [GFA98, vZ02, ABB+02], à l'instar de Clements et Northrop [CN01] évoquent les restrictions apportées lors de la conception du domaine par la prise en considération de styles ou patrons architecturaux (présenté en détails annexe B, section B.3.1).

Voici quelques exemples de structures de l'architecture liées aux modèles de *features*. Dans FODA [KCH+90], la méthode propose une approche en couches pour modéliser l'architecture de la *ligne de produits*. Les niveaux définis permettent de cibler le niveau approprié de réutilisation (architecture du domaine, services du domaine, services communs, et système). FORM [KKL+98] étend l'adoption des *features* à l'espace de la solution et donc de l'architecture (cf. section 3.4.1). Dans Reuse-Driven Software Engineering Business (RSEB) [JGJ97] (et son évolution intégrant les *features* [GFA98]) les niveaux présents sont : système applicatif, spécifique au métier (composants spécifiques au métier), intergiciel (composants fournissant des services indépendants de la plate-forme), système logiciel (infrastructure basse et réseaux). En outre, RSEB [JGJ97] propose de baser l'analyse sur le patron d'architecture "Boundary-Control-Entity". Dans ce patron, les objets de contrôle agissent comme des managers pour interagir avec plusieurs objets d'interface (boundary) et de structure (entity).

4.2.4 Modélisation de l'architecture

Un langage de description d'architecture logicielle est un vocabulaire commun entre les différents acteurs d'un domaine. De nombreux langages de modélisation sont apparus pour permettre la description d'une architecture logicielle et son analyse.

Taylor *et al.* [TMD09] classifient les techniques de modélisation de l'architecture comme suit : (i) les techniques génériques (tel UML, ex. [HNS99, MKMG97]) , (ii) les techniques de description d'architecture primaire (par ex. Medvidovic [MT00] propose une étude de ses approches), (iii) les ADLs spécifiques à un style ou domaine (à l'exemple de Koala [vOvdLKM00] ou AADL [Fei07]), et (iv) les ADLs extensibles (Acme [GMW97]).

Dans les approches *lignes de produits*, l'extension de langage de modélisation générique tel UML est aussi possible [Bos00, ABB+02, Gom04, ZJ06]. Par ailleurs, trois ADLs se retrouvent fréquemment dans la littérature des LdPs, Koala [vOvdLKM00], xADL [DRM00], et AADL [Fei07]. Certaines approches, par exemple [PBL05, LSGF08, HRR+11] combinent des langages d'expression de variabilité avec des langages de description d'architecture spécifiques.

4.2.5 Architecture et évolution et "vice et versa"

La description de l'architecture logicielle, en tant qu'abstraction de la structure de l'application, doit prendre en compte et permettre d'exprimer des évolutions possibles de l'application. [Jaz02] défend que le premier objectif de l'architecture est de guider l'évolution du produit logiciel. La stabilité d'une architecture se mesure à sa faculté à s'accommoder de l'évolution du système sans avoir à réaliser des changements profonds en son sein.

L'architecture doit permettre l'évolution du logiciel mais l'évolution peut également affecter l'architecture. Les raisons et types d'évolution d'une Ligne de Produits sont évoqués dans la section 4.4.

4.3 Ingénierie de l'application et dérivation de produits

La dérivation de produit de la phase de l'*Ingénierie de l'Application* est soutenue par le développement pour la réutilisation (*development for reuse*) de l'*Ingénierie du Domaine*. Cette activité a trait à la réalisation d'un produit singulier par composition, adaptation et spécialisation d'artefacts partagés par la *ligne de produits*, selon un plan de production prédéfini. Cette activité prend le nom de dérivation du produit [ZJ06]. La variabilité explicitée par les modèles de variabilité est résolue pas à pas [CHE04] et les points de variation sont ainsi fixés et liés *(bound)* à des éléments de solution conformément aux contraintes du modèle de variabilité.

Un processus de dérivation efficace pour une organisation permet d'assurer que les efforts requis pour développer la plate-forme des *assets* soient moindres que les bénéfices délivrés au travers l'utilisation des ces artefacts partagés dans la *ligne de produits*. En effet, l'hypothèse sous-jacente à toute *ligne de produits* est que les investissements nécessaires à l'élaboration des *assets* lors de l'ingénierie du domaine doit être conséquente et permettre une dérivation rapide des produits individuels [DSB04]. Cette hypothèse ne tient plus dès lors que la dérivation est inefficiente et donc les gains attendus se retrouvent diminués.

Les outils commerciaux tels *Gears* [Kru07] ou *pure : :variant* [Beu03, Beu08] sont mieux adaptés à la manipulation de fichiers (exigences, code, test, etc), qu'à la gestion de modèles. Dans les *lignes de produits* orientées modèles, les approches existantes se centrent davantage sur la dérivation de modèles structuraux, et la problématique de la composition de modèles comportementaux n'est pas close [IBK11]. De manière générale, deux courants se distinguent, à savoir les approches se focalisant sur les techniques de dérivation et celles se préoccupant du processus de dérivation, par ex. [SLFG09, OMTR10, ROR11].

Du point de vue technologique, les approches de dérivation de produit existantes peuvent être classées en deux types : configuration et transformation.

4.3.1 Dérivation par configuration

La configuration de produit, ou personnalisation de masse, est basée sur le paramétrage et/ou composition de *core assets*. Les variants du produit à réaliser sont sélectionnés en regard des exigences spécifiques et les *core assets* sont automatiquement assemblés. L'approche prend ses origines dans l'idée que cette étape doit être davantage basée sur un paramétrage que sur la manière dont le produit individuel peut être obtenu. Des approches s'apparentant à cette catégorie sont [KCH+90, KKL+98, Kru02, CA05, PKGJ08, HKW08, SWPH09].

Deux activités sont impliquées dans une dérivation de produit par configuration orientée modèles : la configuration elle-même et la réalisation. La configuration consiste en une série de prises de décisions sur les variants pour obtenir les produits désirés sur la base des exigences du client. Un outil de configuration est employé pour assurer cohérence, complétude et conformité de la solution finale. L'activité de réalisation produit les artefacts logiciels et autres extrants comme la documentation par exemple. Classiquement, les artefacts sont sélectionnés dans une base ou "magasin" de *core assets*, ou générés, compilés si la création de nouveaux artefacts est requise, puis le produit est compilé.

L'activité de réalisation est raffinée en deux approches qui prennent en considération (i) la *variabilité positive* (composition de fragments de modèle), ou (ii) la *variabilité négative* (suppression de caractéristiques, propriétés et éléments de modèle à partir d'un modèle global de la

ligne de produits). Des approches, telles CVL [FHMP+09], prennent en charge la combinaison des deux variabilités. Des approches illustrant les variabilités ont été listées dans la section 3.2.3.

4.3.2 Dérivation par transformation

La dérivation de produit par dérivation promeut la transformation de *core assets* et est spécifique à l'IDM. Cette catégorie s'appuie sur les modèles pour représenter les *core assets* et sur des transformations de modèles pour les manipuler et générer, par étapes successives, le produit final. Haugen *et al.* [yHMpOS04] présentent une approche de PLE alignée sur le standard MDA. Mettant en œuvre des transformations successives, le modèle conceptuel initial est finalement implémenté dans la plate-forme cible. D'autres exemples d'approches sont [ZHJ03, ZJ06, Gom04, PBL05, MPL+09, SLFG09, TP08].

4.3.3 Dérivation et flexibilité

Des travaux montrent la rigidité du processus de dérivation et proposent des solutions pour le rendre plus flexible.

Tout d'abord, Deelstra *et al.* [DSB05, DSB04] adoptent une approche ascendante en introduisant leur cadre de *ligne de produits*, qui s'intéresse d'abord à la dérivation avant d'analyser le domaine. Les auteurs présentent un processus de dérivation générique qui implique une dérivation partielle du produit et des itérations additionnelles permettant de répondre aux exigences imposées par le client.

Plus récemment, Perrouin *et al.* [PKGJ08] introduisent une approche de dérivation de produits dirigée par les modèles offrant une plus grande flexibilité. L'approche préconise également de tirer bénéfice de la dérivation partielle : une pré-configuration est réalisée par une sélection des *features* désirées dans un modèle de *features* et une composition automatique des *core assets* correspondants est exécutée (avec l'outil Kompose [FBFG07]). Dans un second temps, le produit pré-configuré est adapté aux besoins spécifiques de l'utilisateur par l'application de primitives, contrôlées par des contraintes (pour ajouter une certaine flexibilité au périmètre de la LdP).

Suivant une approche centrée processus, O'Leary *et al.* [OMTR10] s'inspirent des méthodes agiles et les adaptent à la dérivation de produit : des cycles itératifs sont établis pour les phases de dérivation, test et livraison.

4.4 Évolution d'une *ligne de produits*

D'aucuns s'accordent avec les observations empiriques de Lehman [Leh80], sur la définition de règles concernant les changements des logiciels au cours du temps. L'évolution est indispensable ; la première loi de Lehman est la *Law of Continuing Change*, comprenez par là que le logiciel est assujetti à une évolution permanente pour ne pas devenir obsolète.

4.4.1 Des causes de l'évolution du logiciel

Perry et Wolf [PW92] dissocient deux formes d'évolution, la "dérive de l'architecture" (*architectural drift*) et l'érosion. Une dérive architecturale apparaît lorsque l'architecture existante n'est pas suffisamment comprise par les développeurs et que, conséquemment, les changements réalisés, aussi infimes soient-ils, sont actés à partir d'une base cognitive erronée. L'érosion architecturale est causée par la violation des règles d'architecture.

Une autre perspective sur l'évolution d'un système logiciel est celle du vieillissement [Par94], dans laquelle l'érosion de l'architecture ne constitue qu'une partie [PW92, JLL99, vGB02], l'autre partie étant le glissement du domaine [Par94].

L'érosion de l'architecture signifie que plus des changements sont apportés au système, plus son architecture se dégrade. Ce phénomène s'observe aisément lorsque, suite à des modifications, l'architecture devient instable (entremêlement des fonctionnalités et de la structure) et que les nouvelles modifications sont difficiles à introduire (manque de compréhension de l'existant). Les caractéristiques ou symptômes exhibés par l'érosion sont, selon [JLL99] : une architecture non documentée, des relations entre architecture et code d'implémentation floues, un besoin d'experts métier croissant, des règles d'architecture et de conception non respectées.

Le glissement du domaine implique que le domaine du système, et donc les exigences sur le système logiciel lui-même, changent dans un marché évolutif, à l'exemple de la modification des objectifs des acteurs du projet (ajout ou modification d'exigences), ou de l'environnement du système (par exemple dans les systèmes embarqués, une modification du matériel cible).

Une partie de solution à ces deux problèmes trouve une issue dans la maintenance régulière de l'architecture logicielle ; le succès à long terme de l'architecture est déterminé par sa capacité à survivre à l'évolution. Comme le fait remarquer Parnas à juste titre [Par94], le domaine d'un système logiciel est rarement stable, imposant conséquemment des exigences nouvelles ou modifiées.

Du point de vue maintenance, Swanson [Swa76] souligne l'apparition de trois types de changements : correctif, adaptatif, et perfectif (c.-à-d. respectivement : fixer des *bugs*, s'adapter aux changements du domaine, et augmenter les fonctionnalités du système). En réponse à un changement des exigences et à des opérations de maintenance sur un système, de nouvelles versions de délivrables sont sorties, et de ce fait, le système logiciel évolue. La maintenabilité est la qualité intrinsèque principale pour toute approche d'ingénierie logicielle, et la maintenance (et l'évolution) des logiciels est de loin la phase la plus coûteuse et la plus consommatrice en temps dans le cycle de vie du système [ISO06]. La figure 4.2 rappelle les trois secteurs sources de changements, qui sont : le contexte utilisateur, l'architecture technique et le marketing et les affaires.

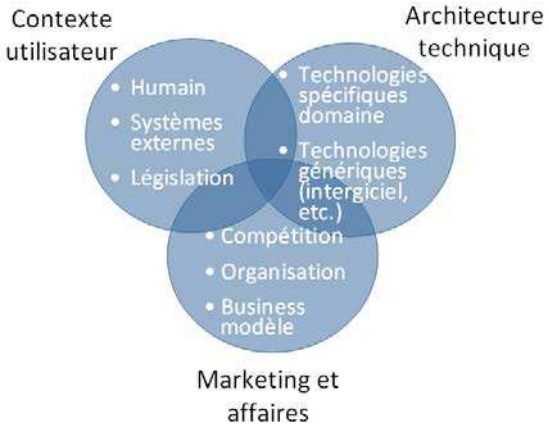


Figure 4.2: Trois secteurs d'évolution.

Il est possible de conclure sur la réflexion suivante : l'évolution n'est pas une hypothèse mais un fait. L'évolution n'est donc en aucun cas synonyme de crise (ou de science extraordinaire) mais est une situation normale de continuité.

4.4.2 De l'évolution des *lignes de produits*

Les *lignes de produits* sont également exposées aux évolutions présentées dans la section précédente, et doivent prendre en compte la maintenance et l'évolution comme un aspect critique au vu de la longévité accrue de nombreux systèmes.

Dans ce cadre, la problématique de co-évolution de l'IDM (évolution d'un métamodèle et co-évolution des modèles, par ex. [CREP08]) est mise à l'écart. C'est une préoccupation à part entière qui est placée hors du périmètre de préoccupation pour se focaliser dans un premier temps sur les modèles, de variabilité et de conception.

La capacité de la *ligne de produits* à évoluer rapidement et fiablement, et d'absorber de nouvelles exigences, est un des défis principaux. Une évolution peut se traduire par l'émergence ou la disparition d'implémentations et concepts, la modification de ceux-ci ou des évolutions algorithmiques. Sur la base de réalisations empiriques, Svahnberg et Bosch [SB99] présentent une taxonomie des types d'évolution affectant la *ligne de produits*.

Taxonomie de l'évolution des *lignes de produits*

La taxonomie de Svahnberg et Bosch est rappelée en annexe B (figure B.3). Elle comprend six catégories d'exigences : construire une nouvelle famille (nouveau marché ciblé), ajouter un nouveau produit (ajout de fonctionnalités, besoins de personnalisation pour un client), améliorer des fonctionnalités (support de nouveaux standards, nouveaux besoins utilisateur), améliorer la conformité à un standard, prendre en compte du nouveau matériel, améliorer les attributs de qualité des plate-formes d'implémentation.

Ces catégories d'exigences impactent l'architecture de plusieurs manières et créent : une séparation en de multiple *lignes de produits*, une dérivation de l'architecture de la LdP (création de branches de développement), une création de composants, des modifications de composants existants, un remplacement de composants, une division de composants (restructuration), une création de nouvelles relations, une modification de relations existantes.

Elle impacte enfin l'architecture des composants eux même, entraînant : la création d'un nouveau cadre d'implémentation, la modification d'un cadre d'implémentation existant, la réduction des fonctionnalités d'un cadre d'implémentation, l'augmentation des fonctionnalités d'un cadre d'implémentation, l'ajout de composants externes.

Évolution de la variabilité

La gestion de la variabilité induit une prise en compte de la maintenance des modèles de variabilité existants, et du peuplement par de nouvelles variabilités (points de variations et/ou variants) ou des suppressions.

L'évolution de la variabilité peut être regardée au travers du cadre de la maintenance [Swa76] et des activités adaptative, perfective et corrective :

- Adaptative : ajout, suppression ou remplacement de variants dans un modèle de variabilité (et des *core assets* correspondants) ;
- Perfective : remplacement ou modification de variants dans le modèle (et des *core assets* correspondants) ;
- Corrective : modification des *core assets* correspondants aux variants, sans impacter le modèle de variabilité

Une taxonomie des différents types d'opérateurs d'évolution pour les lignes de produits avec une teinte exigence est donnée par [SE07]. Les auteurs distinguent trois niveaux d'évolution : exigence (ajout, suppression, modification, avec une distinction pour les trois entre des actions

concernant un produit spécifique, la variabilité ou la commonalité), produit (ajout, suppression), et *ligne de produits* (ajout, suppression, fusion, séparation).

Concernant la modélisation de la variabilité, des études [BPD+10, Ach11] présentent des opérateurs liés à l'évolution de modèles de *features*.

Variabilité et temporalité

Tel qu'énoncé précédemment (cf. 3.1.2) la notion de variabilité dans le temps concerne le génie logiciel et pas uniquement le microcosme de Lignes de Produits. En effet, Elsner *et al.* [EBLSP10] font une synthèse des usages de la variabilité dans le temps et en font apparaître trois types :

- Une variabilité de changements linéaires dans le temps : c.-à-d. relative à la maintenance et à l'évolution. Cette notion a été abordée dans la section précédente.
- Une variabilité ayant trait à différentes versions d'un artefact à un moment donné, c.-à-d. correspondant à de la gestion de configuration. Alors que la plupart des systèmes de gestion de configuration (*software configuration management - SCM*) [Bab86], par ex. [FB03, Loe09, Pil04], cible plus particulièrement les documents textuels tels que le code source, les fichiers binaires, des systèmes de gestion de configuration spécifiques ont vu le jour pour les modèles de l'IDM. Aizenbud-Reshef *et al.* [ARNRSG06a] discutent de l'état de l'art de la traçabilité dans les développements de modèles et des moyens de gestion de configuration. Ducasse *et al.* [DGF05] donnent la définition suivante : une version est un cliché d'une entité à un moment particulier. L'évolution est le processus qui mène d'une version à une autre. L'historique est la réification du concept d'encapsulation de la connaissance sur l'évolution et la version.
- Une variabilité à résoudre dans le temps, c.-à-d. liée à la dérivation de produits dans un contexte de *ligne de produits*. Cette notion a été abordée dans le chapitre 3.

4.4.3 Des sensibilités de gestion de LdP

L'adoption d'une *ligne de produits* est abordée dans plusieurs travaux [CN01, WL99, Bos00, Bos02, SV02], évoquant les degrés de maturité de l'organisation pour la mise en œuvre de la structure de *ligne de produits*. L'adoption est relative à l'élévation systématique d'une organisation d'un état de fonctionnement en projets ou affaires à un état de sophistication plus avancé. Cette phase d'institution est un cas particulier de changement de technologie de par son impact fort sur l'organisation des équipes (raffinement du CONOPS[4] - *concept of operations*, remise en cause du savoir-faire). L'adoption se doit d'être planifiée et de prendre en considération les actifs logiciels existants (*legacy*). Les coûts, risques, et ressources forment classiquement une barrière prohibitive à l'adoption de la *ligne de produits* [Kru02]. Le lancement peut être de type évolutionnaire (sur la base de l'existant) ou révolutionnaire (en faisant table rase de l'existant) selon Simon et Eisenbarth [SE02]. Clements et Krueger [CK02] classent les *lignes de produits* existantes en trois catégories : extractive, réactive ou proactive.

La figure 4.3 issue de [SV02] présente deux graphiques d'investissements, la partie (a) fait référence à l'approche proactive, également nommée "big bang" et la partie (b) fait apparaître l'approche incrémentale réactive.

Proactive

Dans le modèle proactif, une organisation réalise un investissement d'entrée élevé pour développer des éléments réutilisables (architecture, *Core Assets*) et par la suite, sur la base de ces

[4] description haut niveau de la structure organisationnelle et description de la conduite des affaires (processus métier).

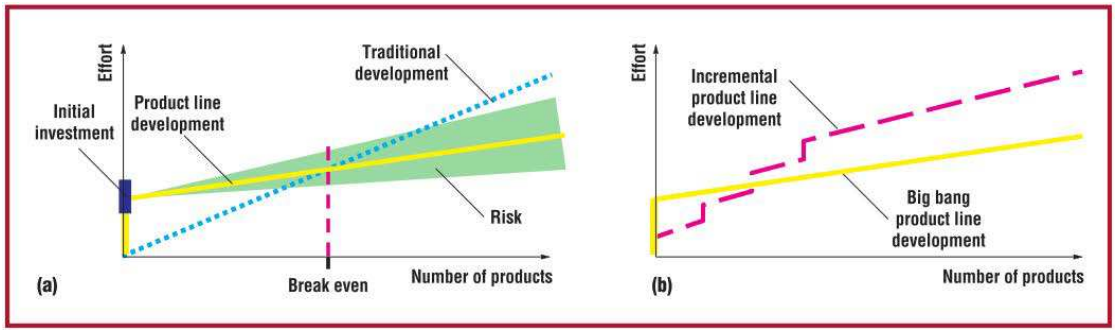


FIGURE 4.3: Graphiques d'investissement - adoption d'une LdP [SV02] : (a) approche proactive, incluant des risques ; (b) proactive vs. réactive.

développements, les produits peuvent être réalisés. Ce type d'approche est effectif pour des *lignes de produits* dans des domaines matures et stables, où les *features* peuvent être prédits. Elle se présente comme l'approche en cascade des développements logiciels conventionnels, à savoir que toutes les variations des produits à un horizon défini sont analysées, incluses dans l'architecture, conçues et implémentées avant toute réalisation de produit. Le modèle proactif peut engendrer des imprécisions dans la variabilité fournie, les *features* prédits pour les futurs produits n'étant pas adaptés et finalement désirés. Le coût d'entrée est conséquent, il est inutile de rappeler que toute réalisation pour la réutilisation voit son coût augmenter (en moyenne 30% de plus pour des composants réutilisables que pour un développement unique [Pou96].

Une LdP configurable ("*configurable software product family*" [CN01]) correspond à une approche proactive totale, où le besoin de développement spécifique pour la réalisation d'un produit est minime voir nul. Les exigences de réalisation du produit sont dans ce cas couvertes en totalité (ou quasi-totalité) par la ligne de produit.

Réactive

Dans le modèle réactif, les éléments réutilisables sont développés au fil du temps quand les opportunités de réutilisations émergent. Cette approche est plus pragmatique et adaptée aux domaines où les *features* ne peuvent être prédits par avance. Bien que cette approche ne nécessite pas d'investissement trop conséquent, le coût de ré-ingénierie et restructuration peut être élevé sans une base architecturale adaptée. L'approche réactive, pour certains aspects, se rapproche d'une approche de développement en spirale ou d'*extreme programming*.

Extractive

Une dernière approche proposée dans [Kru06], appelée modèle extractif, oscille entre le proactif et le réactif. L'approche extractive réutilise un ou plusieurs produits existants pour construire le cadre initial de la Ligne de Produits. Cette approche peut être effective pour une organisation ayant accumulé des expériences et artefacts de développement dans un domaine, et voulant rapidement réaliser une transition des développements conventionnels aux *lignes de produits*. Concernant le retour sur investissement, Krueger va plus loin et défend que l'adoption de la *ligne de produits* logicielle peut être bénéficiaire depuis la première application réalisée, si l'approche est introduite incrémentalement.

L'évolution qui en découle

Bien que la variabilité de la *ligne de produits* et de ses *core assets* anticipe certains changements dans l'espace (différents produits) et dans le temps (différentes versions de produits), tous ne peuvent être prédits et inclus dans la *ligne de produits*.

Les *lignes de produits* adoptées sur un modèle proactif tendent à conserver ce mode et à évoluer de manière proactive également (aucune règle n'impose cet état de fait, et la logique de la première loi de Lehman [Leh80] qui dicte que l'évolution est inévitable, mais difficilement prédictible, suggère plutôt une approche réactive). Dans la pratique, les approches proactives reposent sur un processus rigoureux de planification de l'évolution [MSG96, VDG08, VDJ10, BPD+10].

Les approches réactives sont principalement abordées, soit sous l'angle processus et par l'agilité [HF08, GAM10, ZSP+11], soit par des approches d'implémentation (Clegg *et al.* [CKM02] se reposent sur la fabrique abstraite pour gérer l'incrémentalité). De même, Mende *et al.* [MBKM08] présentent un support outillé à l'approche "*grow-and-prune*"[5] de [FV03] (identification et refactoring de code crée par actions de copier/coller, qui réalise donc une remontée du niveau produit au niveau LdP). Ce type d'approche minimise les risques de développement mais maximise son entropie.

Dans [DSB09], Deelstra *et al.* introduisent une phase d'évaluation de la variabilité pour répondre à l'évolution de celle-ci. Une analyse comparative de la variabilité présente et requise est réalisée dans l'environnement COVAMOF. La remonté d'information depuis l'ingénierie de l'application vers le domaine se trouve également au centre des travaux de Heider [HR10].

De manière générale, l'évolution dans les *lignes de produits* est traitée : (i) par la mise en œuvre de cadres techniques (par ex. [DNGR08b, DNGR08a] basée sur des métamodèles et [DGRN10] autour des différents espaces de modélisations), (ii) en se basant sur de la traçabilité (par ex. : [RP01] permet une gestion des changements, du refactoring et de la reconfiguration ; [AK08] propose des mécanismes couvrant quatre stratégies, l'identification du changement, découverte de l'impact du changement, de la propagation du changement et de la validation de celui-ci), (iii) par des travaux sur la surveillance et la visualisation des changements [TAS07, HRDG09], (iv) par la gestion de version [TM11], (v) par l'analyse de l'évolution plus particulière des modèles de variabilité [DSB09, BPD+10, Liv11], (vi) par le traitement de l'évolution par l'architecture (par ex. [Mac02, DLS05]) et l'évaluation de l'architecture pour apprécier sa modifiabilité (par ex. HoPLAA [OM07] ou EATAM [KKKL08]), (vii) enfin par l'analyse de retours d'expérience [Bos99, Mac02, EDL10].

Pour conclure, il est à souligner que l'évolution incrémentale des *core assets* et de la *ligne de produits* introduit des possibilités d'incohérence devant être gérées convenablement [LHET+10]. Cette vérification peut faire appel à une analyse des impacts liés au changement, qui est définie comme l'identification de conséquences potentielles à un changement, ou à l'estimation de ce qui doit être modifié pour accomplir ce changement [BA96, Boh02].

4.5 Discussion et synthèse

La *ligne de produits* se démarque des développements par projet par une commonalisation d'activités et d'artefacts. Sur le plan du processus, la dualité des ingénieries (Domaine et Application) est caractérisante. Les approches de la littérature font la part belle à l'ingénierie du Domaine, à l'analyse et la modélisation de la variabilité, ainsi qu'à la conception du domaine, en délaissant quelque peu la phase de dérivation des produits. Au delà du processus, des spécificités résident

[5] selon Faust et Verhoef [FV03], la LdP oscille entre (i) une phase de croissance ("grow") libre, qui entraîne un phénomène de mitose logicielle de part l'abondance de clones et de variants, et (ii)) d'élagage ("prune"), qui supprime les branches inutiles.

également dans la gestion de l'architecture des *lignes de produits*. Néanmoins, similairement aux développement d'applications monolithiques, la *ligne de produits* est soumise à des variations externes, l'amenant irrémédiablement (première loi de Lehman [Leh80]) à évoluer : changement d'exigences utilisateurs, d'environnement d'exécution, etc..

Au vu de la grande longévité des systèmes ciblés, une prise en compte de ces changements émergents est primordiale. Dans la confrontation avec ce problème, deux stratégies semblent viables :

- chercher à minimiser la quantité de changements nécessaire pour supporter l'évolution, cette perspective réside dans l'absorption de ceux-ci par l'architecture du système ;
- étudier les possibilités pour minorer la complexité additionnelle requise pour soutenir cette évolution, cette autre voie reposant sur la compréhension et l'explicitation des artefacts manipulés, et sur une gestion de l'évolution, de manière proactive et planifiée, ou réactive.

B 4. *L'évolution des* lignes de produits *requiert une compréhension et donc une explicitation des artefacts manipulés dans le processus, leur positionnement, interactions et dépendances.*

La *ligne de produits* doit traiter l'évolution comme une normalité et non une anomalie [DNGR08b]. L'évolution est inévitable, mais difficilement prédictible, donc assurer un processus de modification favorisant la réactivité est préférable.

Ce chapitre répond à QR 2 et à QR 3, et se faisant couvre la sous interrogation : qu'est ce que l'architecture d'une *ligne de produits* ?

4.5.1 De l'importance de l'architecture des LdPs

Dans les *lignes de produits*, l'architecture assume un rôle dual. La Ligne de Produits en tant que telle possède une architecture (PLA) et les produits résultants également, l'architecture de ces derniers étant obtenue à partir de la PLA et d'une résolution de la variabilité. Face aux changements, il est légitime de se poser la question de savoir si ceux-ci peuvent être implémentés dans le périmètre établi par l'architecture existante, ou si il est inévitable de reconcevoir l'architecture.

Il serait tentant de laisser l'architecture ouverte (qu'elle soit sous-spécifiée) sur les points où la variabilité requise rend possible de standardiser les décisions architecturales. Ce type d'approche a ses désavantages car il devient plus difficile de construire des composants réutilisables et complique la dérivation de produits en transférant les décisions architecturales complexes dans le processus de dérivation.

Par ailleurs, la 2^{e} loi de Lehman [Leh80] pose le besoin de réifier la logique de conception et les règles structurantes de l'architecture. Les patrons d'architecture permettent de maîtriser la complexité logicielle et d'assister le développement, la maintenance et l'évolution de systèmes complexes de grande envergure. Les patrons proposent des principes structurants, qui par là résolvent une part de la variabilité. Ils participent à la remontée et l'échelonnage des décisions de conception dans le nivellement en abstractions.

Sur la base de conceptions existantes et de méthodes d'évaluations, Kazman *et al.* [KBK06] rapportent trois principes concernant l'élaboration d'architecture :

1. une architecture devrait être définie en termes d'éléments de granularité suffisante pour être contrôlée cognitivement par l'humain et assez spécifique pour permettre un raisonnement significatif ;
2. les besoins stratégiques du métier doivent déterminer les exigences des propriétés de qualité ;
3. les exigences des propriétés de qualité guident la conception et l'analyse des architectures logicielles.

Les styles et patrons architecturaux alimentent les architectures avec des stratégies de solution et embrassent ces trois principes. Leur imbrication dans une modélisation globale de la variabilité

améliore la compréhension et le raisonnement. Les patrons d'architecture introduisent un certain degré de prise en compte de l'évolution, et se centrent sur l'évolution qui est susceptible de se produire.

Pour conclure, la recommandation suivante peut être établie :

R 2. *La sélection et l'application d'un patron architectural dédié au domaine sont essentiels pour faire face à l'évolution inhérente des* lignes de produits. *Ce patron doit intégrer la modélisation globale de la* ligne de produits *et donc de son espace de modélisation.*

4.5.2 Du manque de flexibilité de la dérivation

Tout d'abord, il est utile de rappeler que la dérivation de produits, une des activités clés de l'application de l'ingénierie, est peu, voir pas suffisamment, traitée dans les approches existantes [ROR11]. Concernant l'inefficience de la dérivation de produits, un certain nombre de publications relatent les difficultés associées à ce processus. Hotz *et al.* [HGKF03] la décrivent comme lente est sujette à erreur, même si aucun nouveau développement n'est impliqué. Griss [Gri00] identifie la complexité inhérente et la coordination requise dans le processus de dérivation en évoquant que, comme un produit est défini par la sélection d'un groupe de *features*, une mixture complexe et minutieusement coordonnée de divers composants (ou parties de composants) doit être réalisée. Deelstra *et al.* [DSB05] observent que la dérivation de produits individuels à partir de la base commune reste consommatrice en temps et en effort dans de nombreuses organisations. Les efforts à fournir pour accomplir cette tâche ne doivent pas êtres surestimés [Gri00]. Enfin, Perrouin *et al.* [PKGJ08] mettent en lumière le manque de flexibilité des approches de dérivation, permettant d'atteindre un niveau d'efficacité satisfaisant.

De l'étude de la dérivation de produits ressort la recommandation suivante :

R 3. *Le processus de dérivation de produits doit allier automatisme et flexibilité pour une meilleure efficacité.*

CHAPITRE 5

Conclusions et motivations

"Le ſecond [précepte est] de diuiſer chaſcune des difficultez que i'examineraois en autant de parcelles qu'il ſe pourroit , & qu'il ſeroit requis pour les mieux reſoudre."

—René Deſcartes,*Diſcours de la méthode, 1637*

Dans une industrie du logiciel où la complexité intrinsèque des applications ne cesse de croître, un des objectifs du génie logiciel est de conserver l'intelligibilité du processus de construction et de maintenance de ces applications et de le rationaliser.

Cette première partie présente un état de l'art sur la modélisation de la variabilité dans les *lignes de produits* et de son évolution. Le cadre théorique de l'IDM et ses concepts clés y sont abordés (Chapitre 2) et les artefacts manipulés dans le processus de développement des *lignes de produits*, modèles de variabilité et architecture, sont présentés (Chapitres 3 et 4). Ce processus de développement tient sa particularité en une dualité de l'ingénierie du domaine et de l'application. La première partie est une description des notions contextuelles qui sont les prolégomènes de cette thèse, et le présent chapitre en synthétise les observations, recommandations et besoins, pour annoncer les contributions.

5.1 Spécificités des applications

Un dernier aperçu contextuel succinct est présenté pour appréhender au mieux les préoccupations de la thèse, il concerne la caractérisation des applications réalisées par l'entreprise Thales.

Ces systèmes concernent le domaine de la défense, dans lequel la présence du logiciel embarqué est prépondérante. La compétitivité requise pour attaquer ce marché introduit des exigences de budget et de temps de développement restreint, sans pour autant faire de compromis sur les qualités de performances, sécurité et sûreté propres à tous système impliquant des vies humaines. Le nombre de développements réalisés est relativement restreint et sort de la catégorie de la production de masse. Néanmoins de fortes similarités entre les systèmes résultants en termes de fonctionnalités, technologies impliquées, permettent de conclure à l'utilité d'une mise en œuvre de la réutilisation systématique par une approche de *lignes de produits*. La stratégie de déploiement suit logiquement le modèle réactif d'adoption et d'évolution des *lignes de produits*. Il s'agit de

traiter une commonalisation des différentes affaires dans le temps. L'incrémentalité de l'expansion de la Ligne de Produit est primordiale au regard du contexte d'évolution qui suit la première loi de Lehman [Leh80] relative à l'inévitabilité de l'évolution et sa difficulté de prédiction.

En outre, il est inutile de rappeler que les applications du domaine de l'embarqué nécessitent de supporter de fortes contraintes architecturales. Les systèmes à logiciel prépondérant ciblés sont des systèmes de défense antiaériens qui communiquent avec des systèmes externes et sont qualifiés de réactifs [MP92] selon Manna et Pnueli, c.-à-d. qu'ils ont pour finalité de maintenir une interaction avec l'environnement dans lequel ils s'exécutent.

5.2 Synthèse des besoins et recommandations issus de l'état de l'art

Le tableau 5.1 liste les différents besoins et recommandations observés dans cette première partie.

Tableau 5.1: Synthèse des besoins et recommandations résultants de l'état de l'art.

Recommandation 1, p 33	Une modélisation relationnelle doit être spécifique à un domaine métier et à son processus de développement et de contenu sémantique riche.
Besoin 1, p 33	Une modélisation de modèles impliqués dans un processus de développement nécessite une explicitation de l'intention.
Besoin 2, p 59	La modélisation de la variabilité présente un manque de conceptualisation claire des artefacts de variabilité permettant de soutenir une définition possible des relations entre ces artefacts et les autres concepts de modélisation. Celui-ci se doit d'être corrigé.
Besoin 3, p 60	La modélisation de la variabilité affiche un manque d'approche globale de modélisation relationnelle de la variabilité (ergo de la commonalité) dans le contexte de l'IDM, avec une formalisation et une sémantique des relations clairement établies, qu'il convient de pallier.
Besoin 4, p 75	L'évolution des *lignes de produits* requiert une compréhension et donc une explicitation des artefacts manipulés dans le processus, leur positionnement, interactions et dépendances.
Recommandation 2, p 76	La sélection et l'application d'un patron architectural dédié au domaine sont essentielles pour faire face à l'évolution inhérente des *lignes de produits*. Ce patron doit intégrer la modélisation globale de la *ligne de produits* et donc de son espace de modélisation.
Recommandation 3, p 76	Le processus de dérivation de produits doit allier automatisme et flexibilité pour une meilleure efficacité.

La motivation première de la thèse est l'évolution de la *ligne de produits*, qui nécessite une compréhension et modélisation des modèles (B 4) et des espaces de séparations des préoccupations, faisant défaut aux approches de modélisation de la variabilité existantes (dans la conceptualisation de l'espace de modélisation - B 2 -, et dans la formalisation et les usages des relations - B 3 -). Or toute modélisation relationnelle doit être riche de sémantique (R 1) et nécessite une explicitation de l'intention (B 1). En outre, cette compréhension doit s'accompagner d'une organisation et d'une structuration de l'architecture logicielle par un patron dédié au domaine de la *ligne de produits* (R 2). Enfin, aucune gestion de l'évolution n'est possible sans un processus consacré, qui peut se baser sur l'ingénierie de l'application, qui reste le lieu de nombreux efforts de conception et qui nécessite plus de flexibilité (R 3).

5.3 Annonce des axes de recherche

Le problème de recherche auquel cette thèse s'attèle est de faire face à la complexité logicielle, et d'améliorer la réutilisation dans un contexte particulier du logiciel, les *lignes de produits*, qui sont soumises à une évolution permanente. En comparaison des développements de produits monolithiques, l'approche d'organisation globale des LdPs introduit une gestion transverse de variabilité qui est susceptible d'engendrer une complexité accidentelle de gestion indésirable. Dans cette thèse, la compréhension de l'évolution et la gestion appropriée de cette dernière sont considérées comme étant d'une importance vitale. Sur un plan communautaire de l'IDM pour les LdP, il s'agit de renforcer les SoCs et d'améliorer la compréhension de la LdP pour en favoriser l'évolution par un accroissement de la flexibilité de son processus.

Les propriétés défendues par cette thèse pour une mise en œuvre d'une MBPL et une prise en compte de son évolution sont identifiées et définies en trois points clés distincts :

1. Explicitation : explicitation de l'espace de modélisation, des relations et intentions de collaborations entre modèles, ainsi qu'une uniformité de traitement de la variabilité, facilitant de ce fait une compréhension du processus ;
2. Structuration : structuration de la LdP, tant sur le plan de la maîtrise des modèles que concernant l'architecture ;
3. Flexibilité d'évolution : le processus d'évolution doit être clarifié et varié, permettant une aide concrète à l'évolution.

5.4 Annonce des contributions

Contribution Principale *: la contribution de cette thèse est qu'elle améliore la modélisation des LdPs et des activités impliquées pour augmenter leur flexibilité et supporter une évolution ciblée.*

En préambule à toute réflexion, le mémoire propose un état de l'art dont une partie est considérée comme une pré-contribution.

C 0. *Une étude des relations et de leur sémantique et usages dans les approches des LdPs.*

En suivant les axes de recherches identifiés dans la section précédente, les contributions ont trait aux deux ingénieries des *lignes de produits* : modélisation dans l'ingénierie du domaine, et processus dans l'ingénierie du domaine (architecture) et de l'application (dérivation).

Dans la modélisation

S'adressant à la modélisation de la variabilité, la première contribution vise à proposer une séparation des préoccupations générique.

C 1. *Un cadre conceptuel générique de décomposition multidimensionnelle de la variabilité suivant quatre axes.*

En outre, il a été reconnu récemment que le point de vue intentionnel (explicitation du "pourquoi") d'un système facilite la prise de décision. En effet, les décisions sont prises dans le cadre d'un raisonnement reposant sur "ce que l'utilisateur veut" (les besoins/intentions ou buts) et "ce que le système peut faire" (les capacités du système).

C 2. *Une modélisation explicite de l'intentionnalité de l'espace de modélisation des LdPs.*

Cette modélisation est réalisée avec la création du langage PLiMoS (Product Line Modeling Space), défini comme spécialisation du langage de modélisation de [MFBC10]. La figure 5.1 présente une représentation de l'espace de modélisation des *lignes de produits* avec les relations de mégamodélisation. Cet espace se limite dans notre cadre à un unique espace technologique, il ne s'agit pas d'une approche de type écosystème logiciel [Bos09]. Parmi les approches actuelles,

[MFBC10] est celle qui permet une plus grande sensibilité de représentation des relations dans cet espace de modélisation. Dans la figure 5.1, sont représentées les relations de conformité (modèle / métamodèle, et métamodèle / méta-métamodèle), laissant apparaître trois niveaux de modélisation. Les *core assets* sont initialement séparés en modules afin de mettre en œuvre la variabilité positive. L'ensemble de la variabilité de ces *core assets* (présentant des chevauchements sémantiques) est représentée dans le modèle de variabilité (relation *RepresentationOf*) ; un sous-ensemble réalise quant à lui un produit de la LdP (relation *ElementOf*).

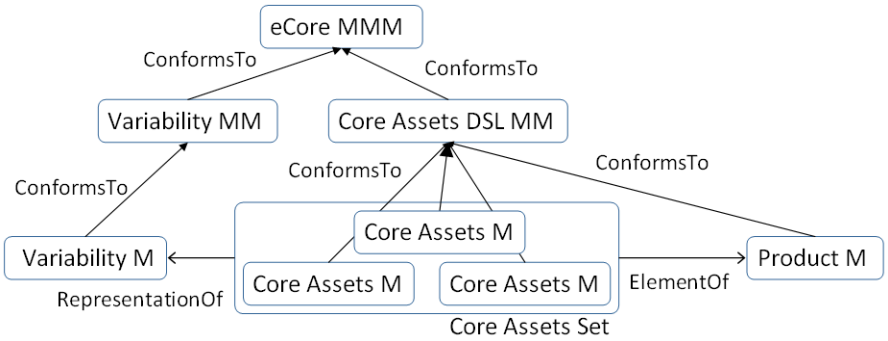


Figure 5.1: Espace mono-technologique de modélisation de la LdP

L'information relationnelle est omniprésente, par conséquent il est essentiel pour les instances qui traitent des connaissances - que ce soient des êtres humains ou des systèmes de traitement de la connaissance - non seulement de classer conceptuellement les entités d'un domaine de discours comme des objets isolés, mais également de décrire les relations entre elles.

L'information relationnelle constitue une part importante de la connaissance des systèmes. Dans une approche de gestion globale des modèles, une modélisation se doit d'adresser différents niveaux de granularité. La position adoptée est donc de découvrir et de décrire les relations intentionnelles nécessaires à une organisation et de spécialiser des relations à plus bas niveau, implémentant ces intentions pour garantir une opérationnalité.

C 3. *Une modélisation des interdépendances et relations entre les éléments de modèles, et de la sémantique liée à leurs usages.*

Dans le processus

Toute modélisation n'est rien sans règles et contraintes, une architecture explicitée à haut niveau permet de bien définir le périmètre de la Ligne de Produits et de faciliter ses activités de dérivation de produits et de maintenance.

C 4. *Une structuration de l'espace de modélisation par la définition d'une plate-forme conceptuelle basée sur un patron pour guider et contraindre la dérivation et l'évolution de la LdP.*

En outre, des approches dénoncent le manque de flexibilité de la dérivation de produits : elle ne permet pas d'absorber les exigences non-anticipées. Par ailleurs cette phase peut être le théâtre de développements spécifiques. La thèse défend que ces développements doivent être, aux besoins, réintégrés d'une manière la plus automatique possible, et ainsi atteindre un haut degré de flexibilité dans la dérivation.

C 5. *Un processus d'évolution incrémental des lignes de produit par extension qui prend son essor dans la dérivation de produits.*

L'approche proposée tend à resserrer les liens entre l'ingénierie de l'application et celle du domaine et par là aborder une évolution incrémentale et réactive de la *ligne de produits*.

Deuxième partie

PLiMoS, une modélisation de l'espace de modélisation des *lignes de produits*

"Truth emerges more readily from error than from confusion."

— Francis Bacon Sr.,*op. cit. The Works of Francis Bacon, 1896*

CHAPITRE 6

Variabilité multidimensionnelle et représentation de l'information relationnelle

"Les classifications ne sont pas [...] *de simples procédures de rangement dans un monde objectivement divisé en catégories évidentes.* [...]*, et la chronique des revirements taxinomiques opérés au cours de l'histoire nous donne l'idée la plus juste des révolutions conceptuelles intervenues dans la pensée humaine."*

—Stephen Jay Gould*, L'éventail du vivant, 1997*

La modularité et la séparation des préoccupations se sont avérées être des solutions efficaces pour faire face à la complexité croissante des développements de *lignes de produits dirigées par les modèles*. Certaines approches proposent des découpages conceptuels des espaces de modélisation de la variabilité et des modèles de conception *(core assets)*. L'approche introduite section 6.1, qui témoigne d'une volonté marquée de généricité, propose un schéma de décomposition qui répond à une problématique d'ingénierie système, permettant une délimitation et une identification claire et systématique des préoccupations et modèles. L'équilibre primordial de cette constellation de modèles repose sur la gestion du corpus de connaissances relationnelles. La seconde section (6.2) propose une réponse au manque d'expression claire et précise de l'information relationnelle, de sa sémantique et de ses usages, par l'introduction d'un langage de modélisation dédié.

Certains éléments de ce chapitre figurent dans les travaux présentés à JLdP 2011 [CMCJ11], et APSEC 2012 [CCJM12b].

6.1 Un schéma générique de séparation des préoccupations pour la variabilité

Très communément dans les développements de *lignes de produits*, le modèle de variabilité est considéré comme représentant l'espace du problème (à l'exemple des approches du type FODA qui reposent sur un modèle de *features* maintenant toutes les combinaisons possibles de produits réalisables) alors que l'espace de la solution est décrit par des graphes de dépendances et des *core assets* liés. D'autre part, dans les approches de développements orientés *features*, par ex. [BSR03], les *features* font partie prenante de la solution. Suivant une approche IDM, il est important de préserver la séparation des préoccupations amenée par la décomposition en niveaux d'abstraction dans l'espace de la variabilité. Le paragraphe présente donc une séparation générique de la variabilité, et son intégration dans le processus de développement, se positionnant dans un cadre multidimensionnel de variabilité.

Dans la suite du mémoire, la séparation des préoccupations est réalisée par une découpe physique en modèles distincts mais peut parfaitement correspondre à une approche multi-points de vue. Le moyen de séparation des facettes exposées dans la thèse n'est pas tributaire de la méthode de sérialisation. Dans la pratique cependant, différents dépôts de modèles sont généralement définis pour les diverses ingénieries, allant jusqu'à appartenir à des espaces réseaux distincts.

6.1.1 Une séparation générique des préoccupations de l'espace de variabilité

À l'instar des approches telles [PBL05, MPH+07, TTBC+09], l'effort est porté sur une dichotomie des espaces du problème et de la solution. Pour compléter cette séparation, une prise en considération de l'espace du contexte environnemental est réalisée (inspirée de [HT08, LK10]). Il en résulte une approche plus systémique et générique que les précédentes.

En effet, la méthode prône une séparation systématique de la variabilité en un triptyque *Contexte (context)*, *Problème (problem)*, *Solution (solution)*. La variabilité de la *ligne de produits* $ModelVar_{LdP}$ est modélisée par ces trois ensembles distincts.

$$ModelVar_{LdP} \triangleq \langle ModelVar_{Context}, ModelVar_{Problem}, ModelVar_{Solution} \rangle$$

Les interactions, chevauchements de ces ensembles ne sont, dans un premier temps, pas abordées, et les sections suivantes s'attachent à décrire la variabilité de chacun de ces espaces.

Variabilité de l'espace du Contexte

La variabilité du contexte environnemental représente la variabilité de l'environnement dans lequel la famille de système est en immersion et est délimitée par deux préoccupations.

1. Tout d'abord, une partie de ce contexte représente la variabilité des acteurs (au sens UML du terme) qui communiquent avec les systèmes issus de la *ligne de produits*. Les acteurs d'un système sont les entités externes à ce système qui interagissent (saisie de données, réception/envoi d'information (capteur, actuateur), etc.) avec lui. Un acteur représente un ensemble cohérent de rôles joués vis-à-vis du système et délimitent le périmètre du système. Ces acteurs permettent de cerner l'interface que le système va devoir offrir à son environnement. Dans une approche de *ligne de produits*, notamment pour des systèmes réactifs, les acteurs peuvent avoir une variabilité. Par exemple, il s'agit de pouvoir dériver différentes applications produits de contrôle de systèmes antiaériens interagissant avec différents types

de radars externes (un à la fois). Dans une gestion du logiciel, il peut s'agir des contraintes sur le matériel ou sur le système d'exploitation par exemple.
2. Une seconde catégorie d'information contenue dans cette variabilité est une information contextuelle relative au métier (représentation des standards à appliquer, des niveaux de confidentialité - par exemple suivant s'il s'agit d'applications délivrées pour la France, l'OTAN, ou d'autres régions du globe, de fiabilité, etc.).

Variabilité de l'espace du Problème

La variabilité de l'espace du problème correspond à une variabilité externe, visible de l'utilisateur. Elle correspond à une variabilité connotée "marketing". Cette variabilité correspond aux missions de la famille de systèmes et ne contient pas d'information sur la structure de réalisation (détails architecturaux ou techniques). Son niveau de précision dépend des exigences client et toutes les informations n'y apparaissent pas. Par exemple, une information de type caractéristique non fonctionnelle comme la performance peut ne pas apparaître telle une variabilité mais comme une exigence externe contraignant le modèle de variabilité de la solution.

Des contraintes provenant de l'espace de la solution peuvent être inférées au besoin pour restreindre l'ensemble des configurations possibles visibles de l'utilisateur. La variabilité externe ne tient pas compte des restrictions techniques imposées par la plate-forme correspondant à la solution d'implémentation.

Cette variabilité peut être liée à la variabilité du contexte et doit être assumée par la variabilité de la solution (idéalement pour une cohérence de la *ligne de produits* : $[\![Var_{problem}]\!] \subseteq [\![Var_{solution}]\!]$, le nombre de décisions ou de choix à résoudre dans l'ensemble des points de variation du modèle de variabilité de la solution doit répondre au besoin exprimé dans le modèle de variabilité du problème).

Il est à noter que dans le cadre d'une *ligne de produits* caractérisée par une évolution réactive, dans certains états d'avancement, la variabilité exprimée par le problème peut être supérieure et non encore satisfaite par la solution existante.

Variabilité de l'espace de la Solution

La variabilité de la solution représente l'abstraction de la variabilité de la solution technique, avec ses points de variations de l'architecture et de l'implémentation. Cette modélisation reflète les choix de conception, et l'organisation (structure hiérarchique) de cette variabilité peut changer profondément de celle de la variabilité du problème. Cette variabilité plus technique est construite par/pour les ingénieurs de conception et propose une réorganisation technique de la variabilité "marketing" de l'espace du problème. La complexité additionnelle apportée par la gestion des variants peut accroître la difficulté des architectes logiciels à concevoir la Ligne de Produits et il est important d'avoir une organisation de la variabilité dédiée à leurs préoccupations. Cette séparation vise à pallier le problème de gestion de la complexité des relations sémantiques à établir entre la vue externe du consommateur qui spécifie ces exigences en termes de propriétés et le producteur qui, quant à lui, tend à spécifier ses systèmes en termes d'éléments de solution (*core assets* pour une *ligne de produits*) : les exigences sont étudiées à la lumière du modèle de variabilité externe (problème) puis interne(solution) avant d'être réalisées par des éléments de conception.

Dans une approche de modélisation par *features*, des *features* composites (abstraits ou concrets) sont généralement introduits pour refléter l'espace technique de la solution (des *core assets*) ; un exemple est présenté dans la section 10.3.

L'organisation hiérarchique intrinsèque de cette variabilité, combinée avec d'autres facteurs, telles que les contraintes d'architecture (qui seront abordées dans le chapitre 10), réduit la complexité cognitive du processus de dérivation de produits.

6.1.2 Une approche multidimensionnelle de la variabilité

Le triptyque de décomposition en Contexte/Problème/Solution présenté dans la section précédente est applicable à différents points de vue et notamment, dans le processus de développement, aux différentes ingénieries : système, logicielle et matérielle.

Projection pour différentes ingénieries

Cette section prend pour cas d'application l'ingénierie logicielle, car les systèmes traités dans le cadre applicatif industriel sont des systèmes à logiciel prépondérant. Il est cependant aisé de translater cette description à l'ingénierie système et matérielle.

La figure 6.1 illustre la vision de l'ingénierie se concentrant sur la variabilité de la conception et des relations avec (i) la variabilité du contexte délivrée par l'ingénierie système, (ii) la variabilité de l'espace du problème utile à la gestion de la *ligne de produits* et aux équipes marketing, et (iii) la variabilité de la solution, interne au développements et propre aux équipes d'ingénierie. La figure fait apparaître des séparations entre trois acteurs (ces dernières ne sont évidemment pas imperméables) et la constitution des ensembles de variabilité, mais avant tout des connexions, qui résultent d'échanges et de compromis entre les différentes parties-prenantes.

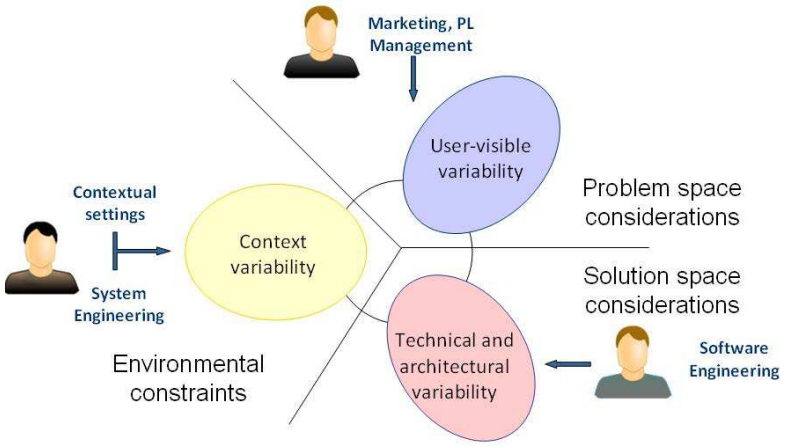


Figure 6.1: Décomposition Contexte - Problème - Solution avec le point de vue de l'ingénierie logicielle

Projection pour l'ensemble des ingénieries

La figure 6.2 représente la découpe contexte / problème / solution appliquée aux différentes ingénieries ainsi qu'une mise en relation. Suivant la démarche hiérarchique systémique, la variabilité externe du point de vue système représente l'union des variabilités des ingénieries logicielle et matérielle.

$$Var_{problem}(System) = Var_{problem}(Software) \cup Var_{problem}(Hardware)$$

La subjectivité induite par les points de vue pose que la variabilité interne de l'ingénierie système corresponde à une part de la variabilité contextuelle pour les ingénieries logicielle et matérielle, une autre part provenant directement de la variabilité du contexte général (soit du contexte système). En effet, s'il est utile de le rappeler, la section 6.1.1 introduit deux plans contextuels ; celle relative au métier peut être transverse aux différentes ingénieries, à l'exemple d'un niveau de confidentialité ou de sécurité qui impacte le développement dans sa globalité.

Considérons le sous ensemble de la variabilité de la solution au niveau système relatif à la définition d'acteurs logiciel et notons $Var_{Solution}(System_{Software}) \subseteq Var_{Solution}(System)$ (égalité en cas de développement logiciel pur) sa variabilité. Par ailleurs, notons $Var_{Context} \subseteq Var_{Context}(System)$ la variabilité du contexte transverse à toutes les ingénieries. La variabilité du contexte logiciel ($Var_{Context}(Software)$), et respectivement celle du matériel ($Var_{Context}(Hardware)$) sont exprimées comme suit :

$$Var_{Context}(Software) = Var_{Solution}(System_{Software}) \cup Var_{Context}$$

$$Var_{Context}(Hardware) = Var_{Solution}(System_{Hardware}) \cup Var_{Context}$$

Les variabilités internes logicielles et matérielles raffinent la variabilité interne système, et l'intersection entre ces ensembles dépend des concepteurs, certaines décisions concernant le logiciel (comme le matériel) pouvant être fixées au niveau système.

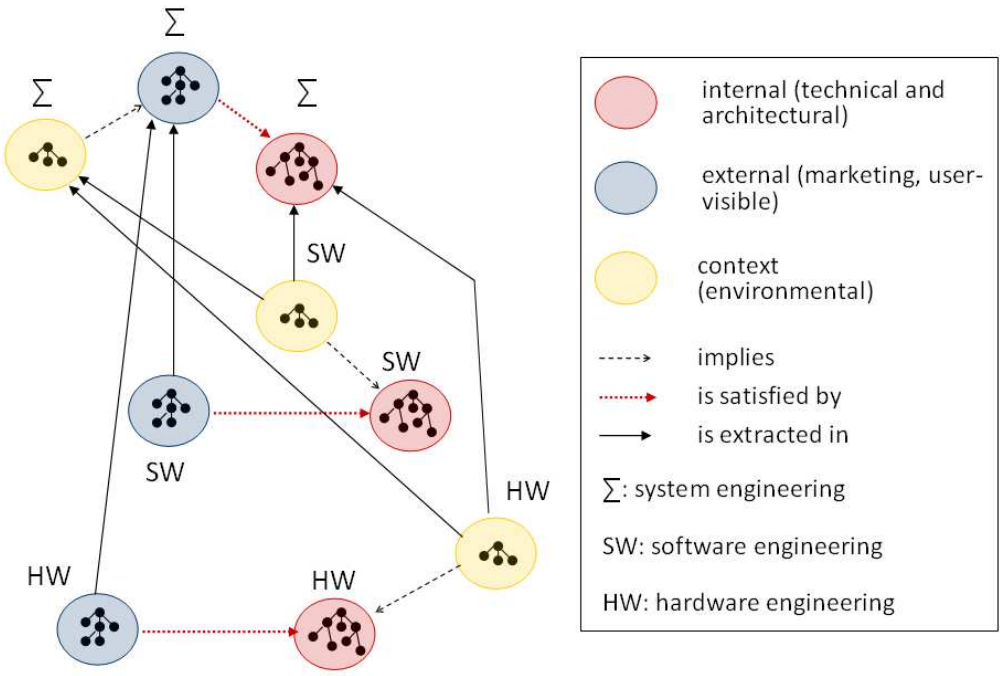


FIGURE 6.2: Décomposition Contexte - Problème - Solution pour les différentes ingénieries

La représentation externe de l'ingénierie système doit posséder une représentation exhaustive de la variabilité utile aux équipes marketing. Il est à noter que concernant le marketing, différentes représentations de la variabilité peuvent exister, notamment pour l'implémentation d'un configurateur pour les clients.
Remarque : les relations d'implication, de satisfaction et d'extraction ne sont volontairement pas décrites dans cette section mais abordées dans les chapitres suivants.

Un cadre conceptuel à quatre dimensions

Cette espace de variabilité est également à positionner en regard de la décomposition en niveaux d'abstraction du processus de développement IDM. Les niveaux d'abstraction présentés dans la section 2.2.1 sont utilisés à titre illustratif dans la suite de la rédaction, soit : *Contexte*[1], *Logique*, et *Physique*. Bien que d'égale importance, la dimension des préoccupations métier est peu ou prou traitée dans les cas d'études exposés dans le document de thèse, les travaux actuels se focalisant principalement sur les trois autres dimensions.

Un cadre générique de décomposition peut être conceptualisé, sur les fondations de dimensions multiples, à savoir :

1D : la dimension de variabilité ;
2D : la dimension du processus de développement (niveaux d'abstraction définis par le processus IDM) ;
3D : la dimension des ingénieries impliquées (système, logicielle, matérielle) ;
4D : la dimension des préoccupations métier (par ex. sûreté de fonctionnement, sécurité, gestion des performances).

La figure 6.3 illustre les quatre dimensions. S'adressant à la modélisation de la variabilité, la première contribution **C 1** vise à proposer une séparation des préoccupations générique ; la première section de ce chapitre a donc introduit un cadre conceptuel générique de décomposition multidimensionnelle de la variabilité suivant quatre axes.

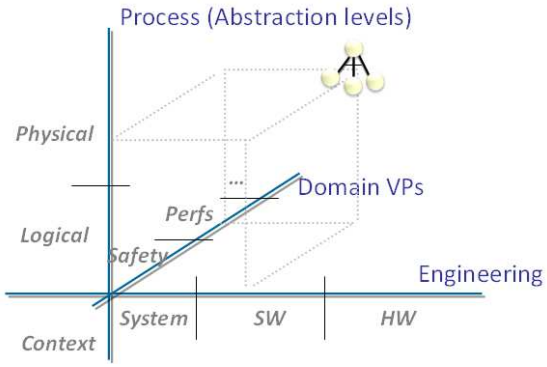


Figure 6.3: Cadre conceptuel de variabilité à quatre dimensions

Une séparation des préoccupations améliore (i) la lisibilité (la variabilité est isolée et associée à un contexte donné), (ii) la compréhension (il est en effet parfaitement raisonnable de considérer que le schéma de pensée des personnes en charge de la gestion de la LdP et du marketing peut différer de celui des concepteurs, architectes et développeurs ; les modèles sont adaptés à leur audience), (iii) l'évolutivité (les différentes variabilités sont amenées à évoluer, leur séparation permet de se préserver d'une influence des unes sur les autres, l'évolution de chacun des modèles gagnant de ce fait en cohérence, la question de l'évolution des relations est décorrélée), (iv) la précision (la granularité de la variabilité est plus ciblée et en adéquation avec les problèmes à traiter, une plus importante quantité d'information se retrouve renseignée).

En outre, l'utilisation de modèles disjoints (i) réduit la complexité visible, et (ii) réaffirme la séparation des préoccupations et des rôles des parties prenantes du processus de développement.

[1]Note : le niveau *Contexte* de la méthodologie ne doit pas être confondu avec la variabilité du contexte environnemental introduite dans les sections précédentes

6.2 PLiMoS, un langage d'expression de l'information relationnelle

La section précédente a présenté un découpage de l'univers du discours de la variabilité des *lignes de produits* suivant quatre axes, la présente se concentre sur l'expression de l'information relationnelle en découlant, la modélisation et l'organisation des relations sémantiques entre les connaissances représentables.

Dans cette thèse, la réponse apportée est typique au génie logiciel, et plus particulièrement à l'IDM, à savoir que l'approche prône l'explicitation des relations, de leur sémantique et de leurs usages, ainsi que la séparation de cette préoccupation de premier ordre, par la création d'un langage dédié (DSML). Le langage se nomme PLiMoS, en référence à l'expression anglaise "*Product Line Modeling Space Language*", la finalité du langage résidant dans la modélisation de l'espace de modélisation des LdPDMs, et donc de la réification du corpus de connaissance relationnelle. Lors de la définition du langage PLiMoS (chapitres 7 et 8), l'accent est placé sur la description de la syntaxe abstraite et de son lien avec le domaine sémantique. La syntaxe concrète du langage iconique, bien qu'entraperçu dans certains exemples le long de ces chapitres, n'est présenté qu'avec la description de l'outillage (chapitre 13).

6.2.1 Problématique et objectif

L'explicitation et la réification des relations participent aux processus de raisonnement et de décisions, processus cognitifs de premier ordre pour la gestion des connaissances. La figure 6.4 illustre la problématique : un plat de spaghetti de relations reliant de manière similaire des préoccupations diverses, et des modèles hétérogènes. La figure 6.5 quand à elle schématise l'objectif et la portée du langage PLiMoS, à savoir une explicitation des relations et de leur motivation, montrant en passant que le langage ne se limite pas à "tracer des traits" entre artefacts de modélisation.

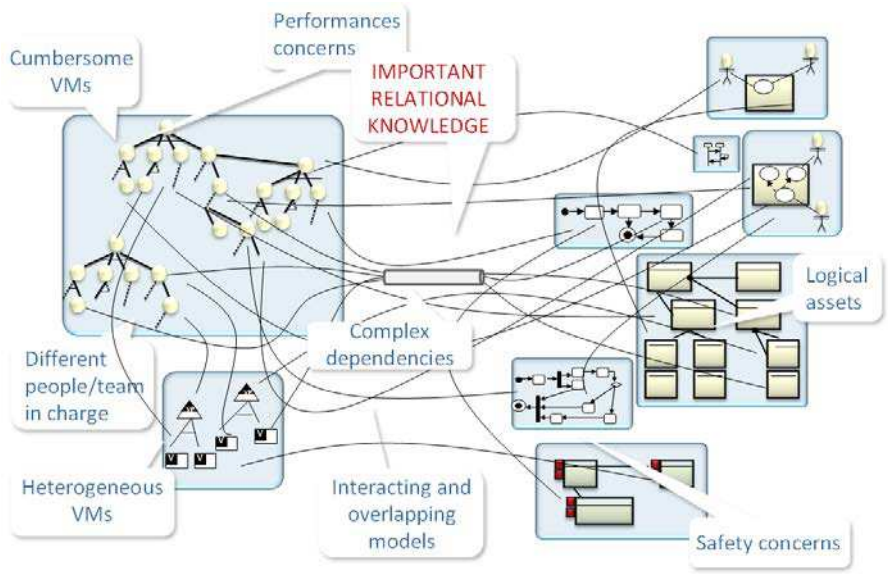


FIGURE 6.4: Espace de modélisation de la LdP, le chaos existe

La démarche se distingue des approches de traçabilité existantes sur plusieurs points. Premièrement, concernant l'angle d'attaque et la direction suivie, qui prend présentement source

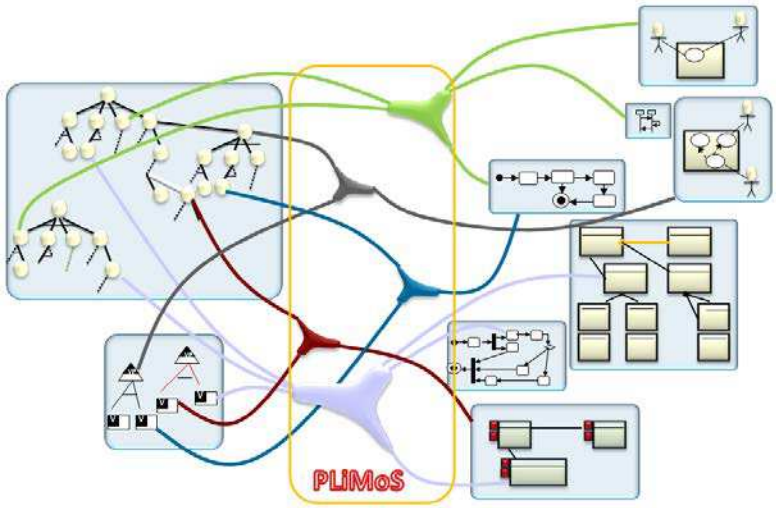


Figure 6.5: Espace de modélisation de la LdP, objectif de PLiMoS

dans le domaine métier, en l'occurrence celui des *lignes de produits*, et non dans l'application de liens à un domaine donné. Deuxièmement, dans les caractéristiques intrinsèques des artefacts considérés, les relations possédant des caractéristiques (par ex. nombre d'intrants et d'extrants, directionnalité, raison d'être, interdépendance) allant au delà, en termes de complexité, des liens dyadiques, classiquement filaires, des approches de traçabilité.

L'objectif est de faire émerger la sémantique nécessaire et suffisante afin que la *ligne de produits* puisse opérer correctement. Cette extraction de la sémantique de l'expertise métier et son injection dans les modèles relationnels permet d'assurer une interopérabilité sémantique entre différents modèles de variabilité et modèles d'architecture de la LdP. La structuration de la sémantique relationnelle par un langage facilite la compréhension de la connaissance et offre un traitement homogène de l'information relationnelle tout au long du cycle de développement.

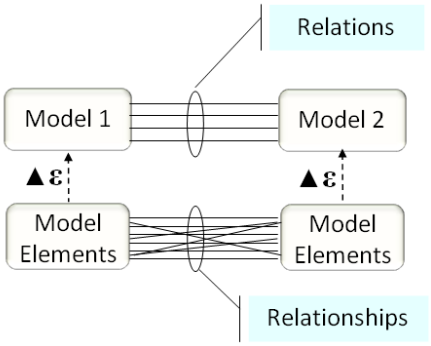


Figure 6.6: Dualité de granularité des relations : modèles et éléments

Les performances cognitives d'un acteur (humain ou informatique) reposent soit sur le champ des représentations auquel il a accès, soit sur la formalisation et réification des concepts essentiels. Dans un univers borné par l'hypothèse d'un espace technologique unique (cf. fig. 5.1, page 80), la démarche proposée s'attarde sur la description de l'aspect relationnel entre les modèles et entre les éléments de ces derniers. La figure 6.6 illustre cette situation de dualité de granularité

relationnelle dans un contexte simple impliquant deux modèles M1 et M2, la réalité ne présentant, cela s'entend, aucune limite théorique quand au nombre de modèles.

6.2.2 L'approche dans sa globalité

Au sein de la communauté IDM, la clarification des concepts manipulés et leur apposition est posée comme facteur clé. Sur la base des travaux de Favre [Fav05a], et plus particulièrement de Muller *et al.* [MFBC10], la relation μ_I (*Representation-Of* intentionnelle) est utilisée dans l'approche pour représenter les relations inter-modèles.

Conceptuellement, le langage PLiMoS ancre ses fondations dans des relations intentionnelles (*Intentional Relations* - IR) qui sont spécialisées par des relations entre éléments de modèles (*relationships*). Ces dernières sont construites sur la base de liens (*Basic Relationhips* - BR), et de propriétés sémantiques associées (*Semantic Properties* - SP). Des fonctions permettent l'association des propriétés sémantiques et des relations intentionnelles aux *relationships*.

$$PLiMoS \triangleq \langle BR, SP, \Phi(SP) \rightarrow BR, IR, \Phi(IR) \rightarrow SP \rangle$$

L'implémentation ECore du langage se base sur les entités suivantes, illustrées par la figure 6.7 : des *RepresentationOf*, éléments de l'ensemble IR ; des *SemanticsInterpretations* constituant l'ensemble SP ; et des *BasicRelationships* (ensemble BR). La réalisation d'un objectif est accomplie par la mise en adéquation de propriétés sémantiques avec une intention, chaque propriété définissant le rôle d'une structure relationnelle basique. Autrement dit, la collaboration définie par la structure et son rôle, suivant une intention donnée, permet l'achèvement d'un objectif.

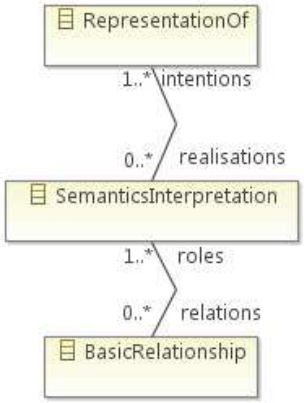


FIGURE 6.7: Entités principales du langage PLiMoS

6.2.3 Des choix de conceptions et d'organisation

La qualité d'un modèle dépend de la nature de la connaissance, mais également de la structuration de celle-ci. Les objectifs de qualité à atteindre se définissent en terme de précision (clarté du langage : sémantique précise, logique de hiérarchisation, définition de niveaux de granularité), de fiabilité (complétude et cohérence : spécialisation pour des espaces de modélisation), et d'extensibilité (généricité et évolution).

Deux critères désirés se retrouvent ici apposés, la généricité et la spécificité (permettant une clarté, une précision, en somme une sémantique formalisée), faisant émerger un semblant de contradiction *in adjecto* par leur antinomicité. Idéalement, tel que le présente la figure 6.8, il paraît intéressant et plus intuitif de définir un niveau de modélisation $PLiMoS_{definition}$ (*Level 1*) générique permettant une spécialisation pour un espace de modélisation donné sur un second niveau $PLiMoS_{specialisation}$ (*Level 2*), offrant un cadre de modélisation $PLiMoS_{implementation}$ (*Level 3*).

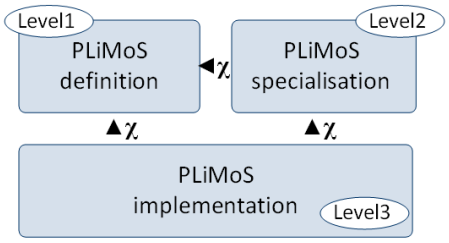


Figure 6.8: Organisation de PLiMoS : solution hiérarchiquement non linéaire

Toutefois, le cadre de métamodélisation du MOF standardisé ($M_{3...0}$, instanciation stricte, hiérarchie linéaire des niveaux de modélisation) ne le permet pas. La première action est de séparer le niveau définition $PLiMoS_{definition}$ en deux préoccupations : $PLiMoS_{specification}$ (pour la spécialisation) et $PLiMoS_{intention}$ (générique, pour la modélisation intentionnelle). Dans un second temps il est à noter que le besoin d'augmentation des capacités de métamodélisation des relations se traduit idéalement par un desideratum de complétion du méta-métamodèle (ce dernier étant relativement modeste dans le cas du MOF et de ECore), cf. figure 6.9 a). En revanche les stratégies d'implémentation existantes permettant de supporter la promotion et la rétrogradation (*demotion*) de concepts de modélisation reposent sur de la transformation de modèles (solution illustrée par la figure 6.9 - b). La solution obtenue s'éloigne d'une extension du niveau de métamodélisation en ce sens qu'aucun objectif de concision n'est retenu, et que la solution recherche l'exhaustivité d'un cadre relationnel avec une perspective associée au domaine des *lignes de produits*.

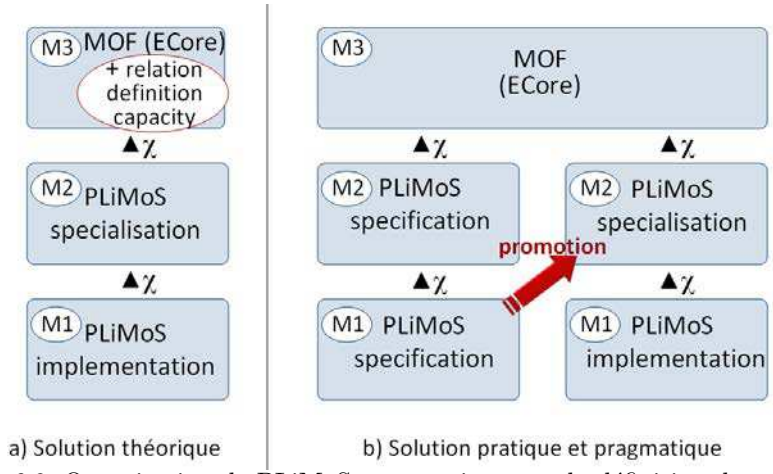


Figure 6.9: Organisation de PLiMoS : promotion pour la définition des relations

En résumé, le langage de modélisation PLiMoS s'organise de la manière suivante (illustration

figure 6.10) : $PLiMoS_{specification}$ est un sous-ensemble du métamodèle générique d'expression de modélisation relationnelle, permettant de paramétrer le sous-ensemble $PLiMoS_{specialisation}$, qui lui est spécifique à un espace de modélisation. Le sous-ensemble $PLiMoS_{intention}$ reste générique et s'attache à proposer un moyen d'expression des relations intentionnelles entre modèles. Ce dernier est obtenu à partir de $PLiMoS_{specification}$ et promu en tant que métamodèle générique et applicable à tout espace de modélisation. Les chapitres suivants apportent des précisions sur les différents sous-ensembles introduits ci-avant.

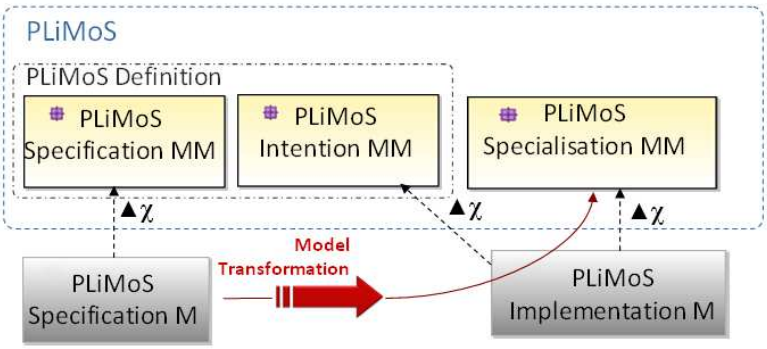


FIGURE 6.10: Organisation de PLiMoS : solution hiérarchiquement linéaire

L'intérêt de réifier les concepts dans un métamodèle spécifique se démontre tant dans l'augmentation de l'intelligibilité de la connaissance traitée, que dans la plus grande facilité de réalisation de l'outillage associé.

6.3 En résumé

En premier lieu, la thèse fournit une description d'un cadre conceptuel générique de décomposition multidimensionnelle de la variabilité répondant à une problématique d'ingénierie système. Ce cadre intègre une approche systémique de la définition de l'espace de modélisation de la *ligne de produits* : application du motif contexte - problème - solution, suivant quatre dimensions (processus, abstraction de modélisation, ingénierie, et préoccupation métier).

En second lieu, l'information relationnelle mise en exergue dans l'espace de modélisation est traitée au moyen d'un langage dédié (DSML) nommé PLiMoS, dont ce chapitre a introduit la construction et la définition. Ce langage, défini dans un cadre MOF, s'organise par une séparation d'un sous-ensemble de définition (spécification et expression de l'intention) et d'un second de spécialisation. Conceptuellement, le langage PLiMoS ancre ses fondations dans des relations intentionnelles (IR), spécialisées par des relations entre éléments de modèles (*relationships*). Ces dernières sont construites sur la base de liens (BR), et de propriétés sémantiques associées (*Semantic Properties* - SP). Des fonctions permettent l'association des propriétés sémantiques et des relations intentionnelles aux *relationships*. Ces notions introduites dans ce chapitre sont détaillées dans les deux suivants.

Le langage PLiMoS propose d'isoler, de clarifier, et d'enrichir la sémantique des relations pour faciliter leur opérationnalité. L'indépendance avec les langages de variabilité et de modélisation apporte une généricité, une interopérabilité entre des modèles hétérogènes, et avantage de ce fait l'évolution de la Ligne de Produits. La structuration du langage offre granularité et abstraction, permettant de renforcer la jonction relations opérationnelles et processus de développement, par des relations intentionnelles de haut niveau.

CHAPITRE 7

Une modélisation intentionnelle de l'espace de modélisation

"Toute connaissance acquise sur la connaissance devient un moyen de connaissance éclairant la connaissance qui a permis de l'acquérir."

—Edgar Morin, *La méthode, t. III, 1986*

Les modèles ne sont en aucun cas établis isolément ; ils existent et répondent à un objectif donné par des parties-prenantes. Dans ce chapitre, l'intérêt est porté sur la capture de la signification intentionnelle des modèles d'une *Ligne de Produits Dirigée par les Modèles*, chose restant communément implicite dans les descriptions. L'approche répond à la nécessité de comprendre les pourquois ("whys") qui sous-tendent les quois et les comments ("whats" and "hows"), les motivations, intentions et raisons sous-jacentes à une modélisation et donc aux activités du processus de développement IDM. Après une introduction des objectifs (section 7.1), du lien entre l'intention et le processus (section 7.2), ainsi qu'une présentation du langage de saisie des relations intentionnelles (section 7.3), une application du langage est proposée dans le cadre multidimensionnel de variabilité présenté dans le chapitre précédent (section 7.4).

Certains éléments de ce chapitre figurent dans les travaux présentés à APSEC 2012 [CCJM12b].

7.1 Objectifs et intention

Dans le cas présent, le grand nombre de modèles introduits par la gestion des *lignes de produits* accroît l'importance de la relation de représentation *"RepresentationOf"*. De plus, dans une approche mono-technologique, elle relègue la relation de conformité au second plan. Les travaux récents de Muller *et al.* [MFB09, MFBC10] s'attachent à fournir un cadre fondateur pour la modélisation et approfondissent la relation de représentation (*"RepresentationOf"* - μ). Les auteurs s'attachent à raffiner la relation et y ajoutent le concept d'intention, concept relevé comme clé dans la modélisation du processus de développement [YM94, Rol98, KC10]. À notre connaissance, et à l'inverse du cadre relationnel de Favre (par ex. [SCFC09]), aucune application pratique des

relations intentionnelles de Muller *et al.* [MFBC10] n'a été définie dans le cadre de l'IDM, et *a fortiori* dans la cadre des LdPs.

Au travers de cette approche, une amélioration de la compréhension de la sémantique des relations entre modèles dans le contexte d'une *ligne de produits dirigée par les modèles* est attendue. Les bénéfices envisagés par l'apport de cette représentation permettent d'adresser la problématique de cohérence de la LdP, de la prise en compte de la maintenance et de l'évolution de la LdP, et enfin, de faciliter la dérivation des produits.

Intention et Objectif sont dans le contexte du langage PLiMoS deux notions complémentaires, car présentant un corrélat de sens ; l'objectif matérialise l'intention, l'intention donne du sens à l'objectif qui a besoin d'être mis en perspective dans une vision globale. L'intention est liée à la motivation première de gestion de l'évolution, l'objectif concerne les capacités visées d'expression de l'intentionnalité dans la modélisation de l'espace de modélisation des *lignes de produits* en vue de cette évolution.

Dans le cadre du langage PLiMoS de modélisation de l'espace de modélisation, les relations intentionnelles entre modèles se matérialisent par des objectifs traduits dans des *relationships* entre éléments de modèles permettant une opérationnalisation de ce dernier.

7.2 Intention et processus de modélisation

L'importance de la notion d'intention dans la modélisation du processus a été rappelée dans la section 2.6.3 en faisant référence notamment aux travaux de Koudri *et al.* [KC10]. À titre d'exemple, la figure 7.1 présente une carte d'intentions en langage MODAL, représentant l'intention de modélisation de l'*External Variability Model* au niveau logiciel. Sont observables sur cet extrait, les concertations à réaliser par les différentes parties-prenantes ; y figurent une description des spécifications requises et le type d'outillage nécessaire. Bien que de grande importance, la thèse ne se concentre pas sur les aspects modélisation du processus, il existe cependant un lien entre cette modélisation du processus et la modélisation des produits, qui passe notamment par les relations intentionnelles.

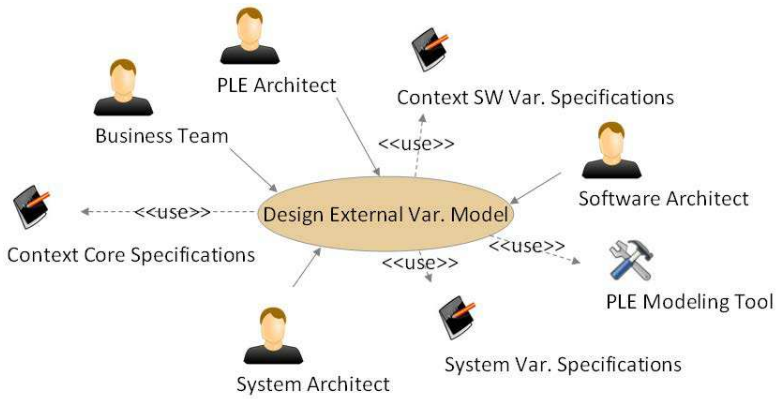


Figure 7.1: Exemple de carte d'intentions, processus de modélisation de l'EVM logiciel

La figure 7.2 positionne la relation intentionnelle μ_I entre modèles en regard de l'intention du processus (I_P). Figure 7.2 b), l'intention du processus I_P est implémentée dans plusieurs processus P, satisfaisant eux-mêmes plusieurs stratégies (S_P). Chaque réalisation du processus (Pe) utilise des artefacts qui ne sont autres que des modèles, et dont les relations entre eux sont

modélisées par des relations μ_I. Un processus manipule une carte de relations intentionnelles μ_I, décrivant les interrelations entre artefacts, cette dernière réalise conséquemment une abstraction des activités du processus.

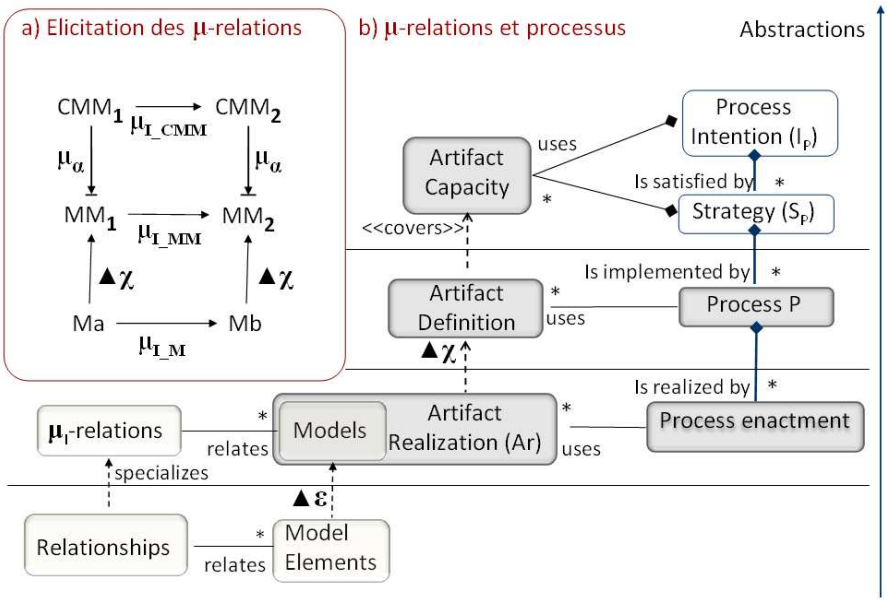


FIGURE 7.2: Relation intentionnelle et intention du processus

La même figure 7.2, partie a) présente un sous-ensemble des concepts de la partie b) mis en relation par deux types de représentation : *consist-of* χ et *representation-of* μ avec sa spécialisation intentionnelle (la flèche pleine ne présente dans ce cas aucun type de relation particulier d'intention, alors que les flèches à la pointe barrée représentent une sous-intention, cf. figure 2.6, page 27). Considérons deux modèles M_a et M_b reliés par une relation μ_{I_M} et respectivement MM_1 et MM_2 les deux métamodèles auxquels ils se conforment. Dans le cadre de DSMLs, il est fortement envisageable d'avoir une cohérence dans la définition du langage et, par conséquent d'avoir une équivalence dans la distance intentionnelle entre les deux types d'artefacts (modèles et métamodèles), soit $\mu_{I\_M} \equiv \mu_{I\_MM}$. D'autre part, la capacité d'un métamodèle est reliée à celui-ci par une relation μ_α caractérisant une sous-intention ; le métamodèle fournit au minimum une implémentation de la capacité. La transitivité de cette relation μ_α permet de poser $\mu_{I\_CMM} \equiv \mu_{I\_MM}$. L'intention du processus I_P couvre l'ensemble des i relations μ_{I_CMM} pour toutes les stratégies, $I_P = \sum \mu_{I\_CMM_i}$. En conséquence, I_P couvre les relations intentionnelles μ_I entre les modèles.

À tout modèle impliqué dans un processus est associé, pour une activité donnée, un rôle, modélisé dans la présente approche par un ensemble de μ-relations situant le modèle dans son environnement.

7.3 Le langage PLiMoS et les intentions

Le langage PLiMoS (pour "*P*roduct *Li*ne *Mo*deling *S*pace language", introduit dans la section 6.2) est proposé pour modéliser l'espace de modélisation des *lignes de produits*. La première partie du langage présenté dans ce tapuscrit est (PLiMoS$_{intention}$), qui permet de définir une modélisation intentionnelle de l'espace. Le langage PLiMoS inclut la relation μ, pour permettre d'améliorer la

compréhension et fournir un cadre formel de définition des préoccupations des modèles. La syntaxe graphique issue de [MFBC10] est préservée dans l'expression du langage PLiMoS (présentées en section 2.3.4, et notamment en figure 2.6, page 27).

Le langage de modélisation PLiMoS$_{intention}$ (Pi) est défini par son domaine syntaxique abstrait $\mathcal{L}_{Pi}$, son domaine sémantique $\mathcal{S}_{Pi}$ et une fonction sémantique $\mathcal{M}_{Pi} : \mathcal{L}_{Pi} \rightarrow \mathcal{S}_{Pi}$ ($[\![\cdot]\!]_{Pi}$).

Définition 14. *Domaine syntaxique $\mathcal{L}_{Pi}$*
Un modèle intentionnel $mi \in \mathcal{L}_{Pi}$ est un 7-uplet $\langle MA, VM, CAM, ROR, RKind, RN, EI\rangle$ tel que :

- *MA est un ensemble non vide et fini d'artefacts de modélisation (Modeling Artefact) ;*
- *$VM \subseteq MA$ est un ensemble non vide et fini de modèles de variabilité (Variability Model) ;*
- *$CAM \subseteq MA$ est un ensemble non vide et fini de modèles de* Core Assets *;*
- *$ROR = BROR \cup CROR$ est l'ensemble fini des relations μ établies entre les artefacts de modélisation ; avec $BROR \subseteq \mathcal{P}(MA)$, ensemble fini des relations μ de Base, et $CROR \subseteq \mathcal{P}(ROR)$, l'ensemble fini des relations Composées μ établies depuis les relations de base (les opérateurs de composition sont l'union, l'intersection, la différence, et la différence symétrique) ;*
- *$RKind : ROR \rightarrow ROREnum$ est la fonction retournant le type de la relation μ ; avec l'énumération $ROREnum = \langle different, shared, subset, superset, same\rangle$, soit l'énumération : différent, partagé, sous-ensemble, sur-ensemble, et identique ;*
- *$RN : ROR \rightarrow RNEnum$ est la fonction retournant la nature de la relation μ ; $RNEnum = \langle \alpha, \gamma\rangle$;*
- *$EI : ROR \rightarrow DI$ est la fonction retournant la description de l'intention (DI) de type chaîne de caractères ;*

En outre, chaque $mi \in \mathcal{L}_{Pi}$ doit satisfaire les règles de bonne formation suivante:

- *Tous les artefacts de modélisation sont reliés entre eux par les relations intentionnelles : $\forall a \in MA(\nexists a' \in MA.\langle a, a'\rangle \notin ROR)$;*
- *Toute relation entre artefacts de modélisation est qualifiée par un type, une nature et une intention : $\forall r \in ROR(\nexists RKind(r) = \emptyset)$; $\forall r \in ROR(\nexists RN(r) = \emptyset)$; $\forall r \in ROR(\nexists EI(r) = \epsilon)$, avec ϵ désignant une chaîne de caractères vide ;*

À noter que le type d'intention "*different*" a un spectre d'applicabilité extrêmement large dans le processus de développement, et que par la suite, seules les relations entre modèles ayant une interaction effective dans le processus sont renseignées.

L'ensemble des combinaisons possibles des modèles et relations d'un espace de modélisation donné est appelé *domaine sémantique*, et est défini comme suit :

Définition 15. *Domaine sémantique $\mathcal{S}_{Pi}$*
$\mathcal{S}_{Pi} \triangleq \mathcal{P}(\langle MA_1, MA_2\rangle_{RKind,RN,EI})$, avec $MA_1, MA_2 \in MA$ et $MA_1 \neq MA_2$, indique que tout modèle syntaxiquement correct doit être interprété tel un modèle de l'espace de modélisation d'une ligne de produits*, c.-à-d. à un ensemble de paires qualifiées (par RKind, RN et EI) d'artefacts de modélisation.*

La fonction liant le domaine syntaxique et les modèles valides correspondants est réalisée par la fonction sémantique.

Définition 16. *Fonction sémantique $[\![mi]\!]_{Pi}$*
$\mathcal{M} : \mathcal{L}_{Pi} \rightarrow \mathcal{S}_{Pi}$ où $\mathcal{M}(mi)$ est l'ensemble des ensembles des paires valides $p \in \mathcal{P}(ROR)$. Chaque $p \in \mathcal{M}(mi)$ est telle qu'elle:

- *est qualifiée par un type : $RKind(p) \neq \emptyset$;*
- *est qualifiée par une nature : $RN(p) \neq \emptyset$;*
- *est qualifiée par une description d'intention : $EI(p) \neq \epsilon$;*

De plus chaque artefact de modélisation doit être relié par une p.

La figure 7.3 représente un extrait du métamodèle PLiMoS, décrivant les relations inter-modèles entre artefacts. La relation "RepresentationOf" (μ) relie des artefacts de modélisation composant l'espace de modélisation de la *ligne de produits dirigée par les modèles* : modèles de variabilité et de *core assets*.

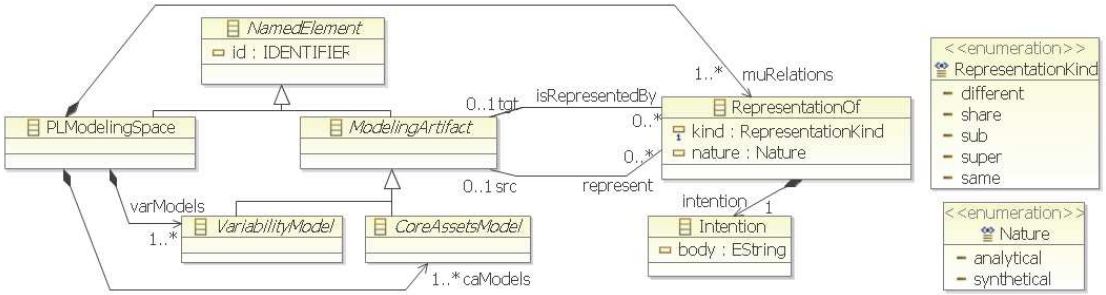


FIGURE 7.3: Extrait du métamodèle PLiMoS - relations intentionnelles entre artefacts de modélisation

En outre, et contrairement à l'approche d'origine, une relation intentionnelle peut être définie en ayant pour source et cible un même modèle, dénotant de ce fait l'encapsulation d'une multiplicité de considérations de modélisation dans un même modèle. Cela est réalisé dans la perspective d'expression de relations intra-modèles spécialisant diverses intentions (cf. chapitre suivant, 8).

La figure 7.4 fournit des détails sur les relations μ composées. Une relation composée est obtenue par composition de relations μ existantes (basiques ou composites) à l'aide des opérateurs ensemblistes suivants :

- *Union* ($\cup$) : l'union (propriétés d'associativité, de commutativité, d'idempotence, etc.) est associée à l'opérateur logique *ou inclusif* ;
- *Intersection* ($\cap$) : l'intersection (associativité, commutativité, d'idempotence, distributivité sur l'union, etc.) est associée à l'opérateur logique *et* ;
- *Difference* ($\setminus$) : la différence fournit le complément absolu et relatif ($A \setminus B = x \in A | x \notin B$) ;
- *Symmetrical Difference* (Δ) : la différence symétrique est associée à l'opérateur logique *ou exclusif* ($A \Delta B = (A \cup B) \setminus (A \cap B)$) ;

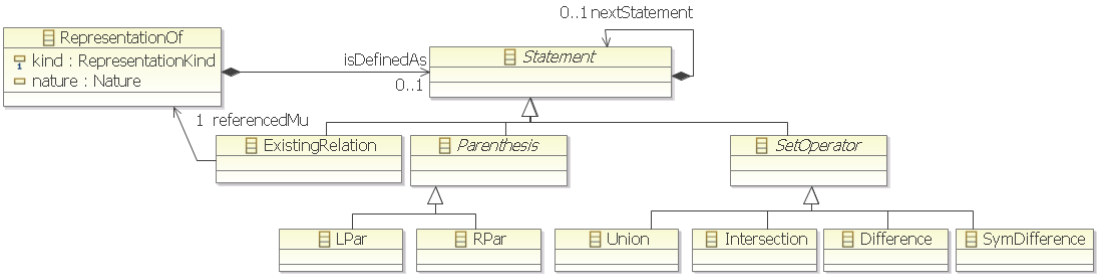


FIGURE 7.4: Extrait du métamodèle PLiMoS - détails des relations intentionnelles composées

L'expression résultante de l'association des déclarations doit être bien formée. Des contraintes OCL (cf. listing 7.1) permettent de s'assurer (i) d'un nombre équivalent de parenthèses ouvrantes et fermantes, (ii) que l'ensemble des *statements* contient une référence vers une relation μ existante, (iii) que chaque référence est précédée et suivie d'un opérateur ou d'une parenthèse.

```
context : RepresentationOf
inv : self.isDefinedAs->closure(s : Statement |
s.nextStatement)->including(self.isDefinedAs)->selectByType(LPar)->size() =
     self.isDefinedAs->closure(s : Statement |
s.nextStatement)->including(self.isDefinedAs)->selectByType(RPar)->size()

inv : self.isDefinedAs->closure(s : Statement |
s.nextStatement)->including(self.isDefinedAs)->selectByType(
    ExistingRelation)
->notEmpty()

inv : self.isDefinedAs->closure(s : Statement |
s.nextStatement)->including(self.isDefinedAs)->selectByType(
ExistingRelation).oclContainer()->selectByType(ExistingRelation)->isEmpty()
and self.isDefinedAs->closure(s : Statement |
s.nextStatement)->including(self.isDefinedAs)->selectByType(
ExistingRelation).oclContents()->selectByType(ExistingRelation)->isEmpty()
```

Listing 7.1: Expressions OCL - contraintes sur le langage ensembliste

En outre, une autre contrainte OCL, exprimée par le listing 7.2 permet de différencier les relations basiques (entre éléments de modèles) et composées (composition de relations μ existantes), en s'assurant que l'entité *RepresentationOf* possède soit des références vers des éléments *src* et *tgt*, soit une référence *isDefinedAs*.

```
context : RepresentationOf
inv :   if self.isDefinedAs->notEmpty()
        then src->isEmpty() and tgt->isEmpty()
        else src->notEmpty() and tgt->notEmpty()
        endif
```

Listing 7.2: Contrainte OCL - relations basiques ou composées

La vérification de bonne formation globale et de conformité est dissociée des contraintes structurelles et réalisées dans l'outillage associé par l'évaluation d'une translation de l'expression dans le langage OCL (opérateurs : *union*, *intersection*, -, *symmetricDifference*). Les axiomes de composition des intentions suivent les règles définies par Muller *et al.* [MFBC10], représentés dans ce document dans la figure B.1, page 213.

7.4 Application dans l'espace de variabilité à quatre dimensions

Dans cette section, la modélisation des relations est renseignée par une syntaxe concrète identique aux travaux de Muller *et al.* (pour rappel, figure 2.6, page 27). À des fins d'illustration, une application au cadre générique de variabilité multidimensionnelle (présenté section 6.1) est réalisée, représentant un sous ensemble de relations pertinentes de cet espace.

7.4.1 Abstraction de la variabilité

L'intention est de modéliser une variabilité existante ; les modèles de variabilité réalisent une abstraction de l'espace de conception défini par un ensemble de core assets, peu importe le niveau d'abstraction ciblé. Dans la relation intentionnelle $\mu\#1$, le modèle de variabilité *VM* abstrait la variabilité de l'ensemble *CASet* (relation analytique), et l'intention de modélisation de *VM* couvre partiellement celle de *CASet*. Le modèle de variabilité doit être correct par rapport à la modélisation de la conception.

$$VM \underset{\mu\alpha}{\longrightarrow\!\!|} CASet \qquad \mu\#1$$

Dans une perspective d'ingénierie applicative, l'intention est portée sur l'obtention d'un produit sur la base des modèles existants. Le processus qui en découle inverse la nature de la relation qui en devient synthétique. En effet, le modèle de variabilité comporte également la préoccupation de configuration, et la relation $\mu\#2$ présente le lien entre ce modèle et le modèle du produit ("Product Model" - *PM*). Le produit doit être valide par rapport au modèle de variabilité (contraintes de la LdP et de la configuration).

$$VM \underset{\mu\gamma}{\longrightarrow\!\!|} PM \qquad \mu\#2$$

7.4.2 Hiérarchisation des niveaux d'abstraction du processus

Le processus pris en exemple définit les niveaux d'abstraction suivants : *Contexte*, *Logique* et *Physique*. L'intention couverte est de modéliser un espace de modélisation distinguant une variabilité multi-niveaux. La figure 7.5 représente deux espaces, celui de la variabilité et celui de la conception dans lesquelles sont représentés, respectivement, les modèles de variabilité ("Context Level Variability", "Logical Level Model" et "Physical Level Variability") et les modèles de *Core Assets* ("Context Core Assets", "Logical Core Assets" et "Physical Core Assets"). Pour une question de lisibilité, la figure 7.5 ne présente qu'un extrait des relations intentionnelles. Les relations non représentées et non décrites ci-après sont des relations dénotant une séparation dans les intentions de modélisation (de types différents $\dashrightarrow$).

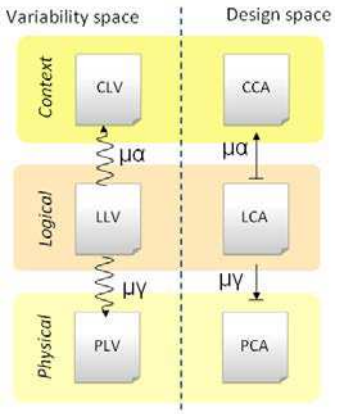


FIGURE 7.5: Modélisation PLiMoS - quelques relations intentionnelles dans le processus IDM

Dans l'espace de la variabilité

Entre la variabilité du niveau Contexte *et les autres variabilités.*

La variabilité du niveau *Contexte* correspond à une variabilité de l'espace du problème, en regard avec les *core assets* du contexte. Les deux autres variabilités (LLV et PLV) correspondent à l'expression de la variabilité de la solution, à deux niveaux d'abstraction différents.

La variabilité CLV doit être assumée par la variabilité de la solution, une partie au niveau *logique*, une autre au niveau *physique* (relation synthétique de dérivation). Dans la relation $\mu\#3$, les deux modèles partagent une même intention, à savoir la modélisation de la variabilité du système. Le LLV se doit d'être correct par rapport au CLV, ce qui signifie que chaque variabilité exposée à l'utilisateur et correspondant à une variabilité logique devra être satisfaite par le LLV.

$$CLV \xleftarrow[\mu\alpha]{} LLV \qquad \mu\#3$$

La relation $\mu\#4$ met également en relation la variabilité du problème (CLV), mais cette fois avec celle de la solution de conception physique (PLV). Le PLV modélise la variabilité au niveau *Physique* et, comme énoncé précédemment pour la variabilité logique, le modèle se doit d'être correct par rapport au CLV. Ceci signifie que chaque variabilité exposée à l'utilisateur et correspondant à de la variabilité physique se doit d'être satisfaite par le PLV.

$$CLV \xleftarrow[\mu\alpha]{} PLV \qquad \mu\#4$$

À noter que cette relation $\mu\#4$ peut être considérée comme composée des relations $\mu\#3$ et $\mu\#5$, respectant les axiomes de compositionalité de la figure B.1, page 213.

Entre la variabilité du niveau Logique *et* Physique *:*

Concernant la variabilité logique et physique ($\mu\#5$) : Les modèles en vis-à-vis (c.-à-d. LCA et PCA) sont intentionnellement liés (raffinement de conception), et les LLV et PLV le sont également, la variabilité étant complétée par le raffinement (la relation est synthétique, en raison de la temporalité de la relation).

$$PLV \xleftarrow[\mu\gamma]{} LLV \qquad \mu\#5$$

Dans l'espace de la conception

Les modèles de description du contexte (CCA) reflètent les spécifications, et sont réalisés par les modèles logique et physique (respectivement appelé LCA et PCA). Les modèles physiques sont des raffinements des modèles logiques. Dans la $\mu\#6$, le CCA décrit l'espace du problème et le LCA la solution de conception logique. La relation étendue $\mu\alpha$ exprime le fait que le LCA est plus riche que le CCA en termes d'intention, car il contient des détails de conception additionnels. Une relation similaire est établie entre le CCA et le PCA.

$$CCA \xleftarrow[\mu\alpha]{} LCA \qquad \mu\#6$$

Dans la $\mu\#7$, le PCA, qui est spécifique à la plate-forme, est un raffinement du LCA (logique et donc plate-forme indépendant) qui peut être obtenu par transformation (la nature de la relation est synthétique).

$$LCA \xrightarrow[\mu\gamma]{} PCA \qquad \mu\#7$$

7.4.3 Triptyque contexte - problème - solution

Les modèles considérés ont été présentés dans la section 6.1.1 et sont de type (i) modèle de variabilité de l'environnement contextuel ("Context Variability Model" - CVM, toujours à ne pas confondre avec le modèle de niveau *Contexte* - CLV), (ii) modèle de variabilité de l'espace du problème (qualifié de modèle externe - EVM), et (iii) modèle de variabilité de la solution (ou dit interne - IVM). La figure 7.6 illustre le cas particulier de l'ingénierie logicielle. L'intention de modélisation se retrouve dans la perspective de réalisation d'un produit.

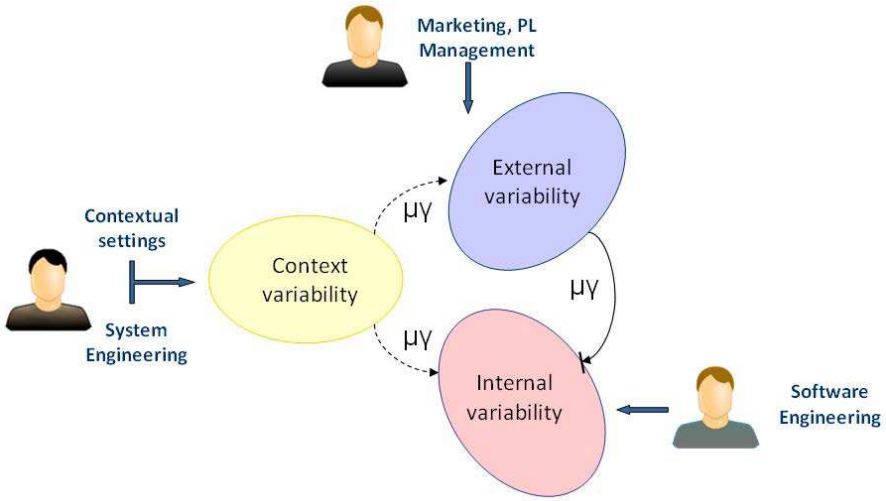


FIGURE 7.6: Modélisation PLiMoS - relations μ de décomposition contexte - problème - solution dans le cas de l'ingénierie logicielle

Entre le CVM et les deux autres types de modèles

Le contenu du *CVM* est établi, pour majeure partie, à la suite de discussions entre les ingénieurs systèmes et logiciels. Lors de l'étape de dérivation dans l'ingénierie de l'application, les variants du *CVM* sont sélectionnés pour guider la réalisation du produit en respectant les contraintes du contexte. Une sélection depuis le contexte facilite la configuration, dans le sens où elle garantit un respect et une validation par construction des contraintes issues du contexte : les *variants* externes sont induits par ceux du contexte environnemental. Dans la μ#8, bien que l'impact du *CVM* sur l'*EVM* soit un point clé lors du processus de dérivation, les modèles sont conçus avec des intentions totalement indépendantes en tête. Cette relation μ indique un décalage entre deux points de vue. Une relation identique peut être définie entre le *CVM* et le *IVM*.

$$CVM \xrightarrow[\mu\gamma]{} EVM\,; \quad CVM \xrightarrow[\mu\gamma]{} IVM \qquad \mu\#8$$

Entre l'EVM et l'IVM

L'*EVM* est employé pour discuter du problème des exigences entre deux principaux acteurs, les ingénieurs pour l'aspect développement, et les affaires pour les négociations. Les relations entre les points de variation de l'EVM et de l'IVM résultent des négociations entre les affaires, les ingénieurs LdPs et les ingénieurs logiciels.

Tel qu'évoqué précédemment, les variants de l'espace du problème doivent être *satisfaits par* des variants internes "concrets" ; à l'inverse, certains variants internes sont susceptibles de ne pas être montrés comme tel au client.

Les relations $\mu\#3$ et $\mu\#4$ traitent du positionnement du modèle de variabilité du problème avec deux modèles de la solution à deux niveaux d'abstraction. La prise en compte de la variabilité des espaces du problème et de la solution dans leur globalité, c.-à-d. l'EVM et de l'IVM, résulte dans la $\mu\#9$ (pour les ingénieries terminales telles que celle du logiciel et du matériel). Chaque variant de l'EVM doit être exaucé par la variabilité de la solution. L'intention de l'EVM est supplantée par celle de l'IVM avec la satisfaction de la variabilité.

$$EVM \xrightarrow[\mu\gamma]{} IVM \qquad \mu\#9$$

7.4.4 Ingénieries système, logicielle, matérielle

Toujours suivant la même intention de réalisation d'un produit, la figure 7.7 représente une partie des relations intentionnelles (pour des raisons de visualisation) reliant les différentes ingénieries du processus. Ces relations ont été exprimées dans les sections précédentes et sont uniquement illustrées dans la figure. L'ingénierie système n'étant pas terminale, l'ensemble de la variabilité externe n'est pas forcément couverte par le modèle interne, mais peut également l'être par les variabilités spécifiques du logiciel et du matériel.

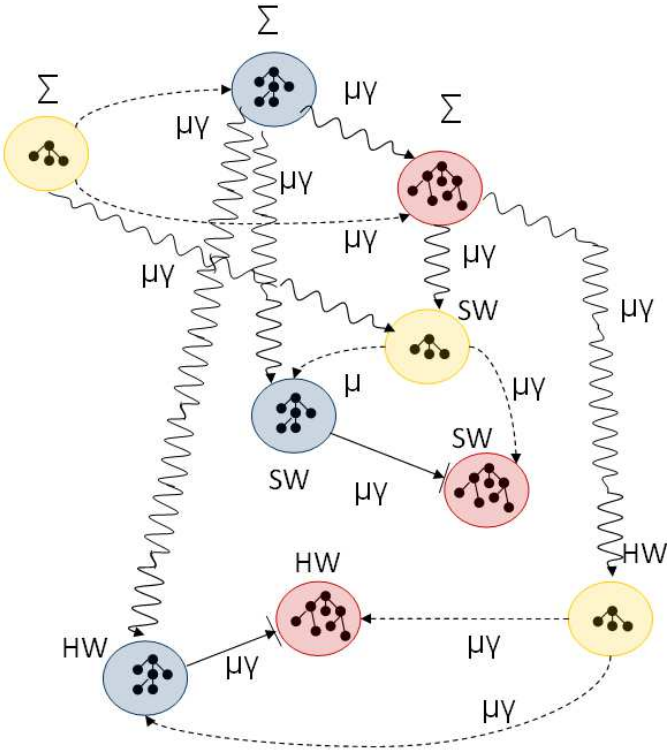


Figure 7.7: Des relations *mu* pour les différentes ingénieries

7.4.5 Un focus sur l'espace de conception

Cette section décrit des relations de modélisation de l'espace de conception, plus particulièrement centrées sur une approche de modélisation avec le langage UML.

Use-Cases et diagrammes de séquences.

Au niveau contexte du système, le *Use-Case* décrit un ensemble de relations et dépendances entre des groupes de cas d'utilisation et des acteurs participant au processus. La description comportementale d'un cas d'utilisation (*Use-Case*) peut être réalisée, entre autre, au moyen de diagrammes d'interaction (diagrammes de séquences, ou de collaboration). Le diagramme de séquence propose une description d'un scénario d'échange de message décrivant plus finement des relations du *Use-Case*. Les deux diagrammes sont de niveau instance, et le diagramme de séquence (SD) étend l'intention de modélisation du *Use-Case* (UC), déterminant la relation $\mu\#10$.

$$UC \xrightarrow[\mu\alpha]{} SD \qquad \mu\#10$$

Diagrammes d'état et de séquences.

À un niveau logique par exemple, considérons maintenant l'interaction entre le diagramme d'état (StD) et le diagramme de séquence (SD). Dans une approche exécutive, il est envisageable de générer le diagramme de séquence depuis une spécification des interactions dans une machine à état (relation $\mu\#11$).

$$SD \xleftarrow[\mu\gamma]{} StD \qquad \mu\#11$$

À l'inverse, dans une démarche de modélisation intentionnelle par extension, la machine à état est obtenue à partir d'un ensemble de scenarii (relation $\mu\#12$).

$$\bigcup SDs \xrightarrow[\mu\gamma]{} StD \qquad \mu\#12$$

Diagrammes de classes et d'état.

Le diagramme de classes (*Class Diagram* - CD) décrit la structure d'un système. Sur la base de cette structure, le comportement peut être exprimé par un diagramme de machine à état. Ces deux modèles partagent l'intention de modélisation d'un même sujet en y abordant deux aspects de corrélation forte, le statique et le dynamique. La relation $\mu\#13$ dénote une différence de point de vue.

$$StD \xleftarrow[\mu\alpha]{} CD \qquad \mu\#13$$

7.4.6 Des considérations métiers.

Cette section illustre le cas de la dernière dimension de l'approche (cf. section 6.1.2), celle de la préoccupation métier. Suivant l'approche de Voirin [Voi08], et sur la base d'une décomposition structurelle fonctionnelle, les différents points de vue métier sont observés.

La figure 7.8 montre un cas d'application des relations entre la structure de base, une modélisation de la sûreté de fonctionnement (SdF), et de la sécurité, ainsi que les deux points de vue résultants (respectivement VP_{SdF} et $VP_{Securite}$). Ces points de vue étendent la structure de base par les concepts métiers ciblés. Quelques relations de différence d'intention sont représentées dans la figure pour représenter l'orthogonalité de certaines informations (la représentation exhaustive de ces dernières alourdirait considérablement la représentation).

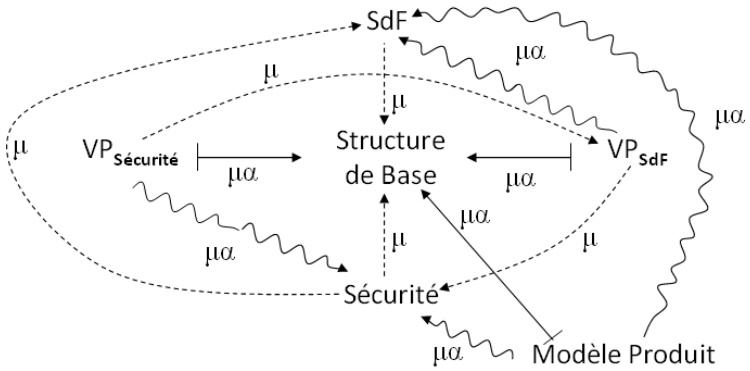


Figure 7.8: Des relations *mu* explicitées pour l'approche ARCADIA [Voi08]

7.5 En résumé

Le point de vue intentionnel par l'explicitation du "pourquoi" d'un système facilite la prise de décision, à savoir que ces décisions sont prises dans le cadre d'un raisonnement reposant sur les besoins, intentions et buts), et sur les capacités du système. L'approche répond à la nécessité de comprendre les pourquoi qui sous-tendent les quoi et les comments, les motivations, intentions et raisons sous-jacentes à une modélisation et donc aux activités du processus de tout développement IDM.

Le langage PLiMoS et son sous-ensemble intentionnel repose principalement sur la relation μ, cette dernière est introduite et mise en regard du processus de modélisation et de son intention. La syntaxe et la sémantique du langage PliMoS$_{definition}$ sont formalisées et implémentées dans un métamodèle outillé. Le sous-ensemble PliMoS$_{intention}$ n'est pas spécifique aux *lignes de produits* mais son applicabilité se vérifie dans de nombreux exemples d'application dans l'espace de modélisation des *lignes de produits*.

Avec PLiMoS, les relations intentionnelles entre modèles se matérialisent par des objectifs traduits dans des *relationships* entre éléments de modèles permettant une opérationnalisation de ce dernier ; ces deux points sont détaillés dans les deux chapitres suivants.

CHAPITRE 8

Méta et modélisation des relations opérationnelles

"Penser sans jamais clore les concepts, briser les sphères closes, rétablir les articulations entre ce qui est disjoint, essayer de comprendre la multidimensionnalité, de penser avec la singularité, avec la localité, avec la temporalité [...]*."*

—Edgar Morin, *Les théorie de la complexité, colloque de Cerisg, 1991*

Dans le cadre du langage PLiMoS de modélisation de l'espace de modélisation, les relations intentionnelles entre modèles se matérialisent par des objectifs traduits dans des *relationships* entre éléments de modèles permettant une opérationnalisation du langage. L'articulation entre les diverses parties prenantes se spécialise à un niveau de granularité plus fin.

Ce chapitre s'intéresse à une méta-modélisation réalisée à partir d'un paramétrage depuis $\text{PLiMoS}_{specification}$, permettant d'établir un modèle, paramètre de la génération du métamodèle de spécialisation. $\text{PLiMoS}_{specification}$ est un langage générique de définition de modèles relationnels, qui laisse apparaître néanmoins une certaine spécialisation aux *lignes de produits* afin de proposer certaines facilités d'utilisation. Le langage permet la modélisation d'un large panel de relations, notamment de relations de granularité plus fine de l'espace de modélisation des LdPs, et répond aux intentions relevées avec le langage $\text{PLiMoS}_{intention}$ dans le chapitre précédent. Ce faisant, le chapitre aborde également à titre illustratif la définition de $\text{PLiMoS}_{intention}$ à partir de $\text{PLiMoS}_{specification}$. Le chapitre suivant (chapitre 9) traite du *comment* et donc des activités et opérateurs permettant une opérationnalisation du langage, couvrant notamment la description des règles de génération du langage $\text{PLiMoS}_{specialisation}$.

Certains éléments de ce chapitre figurent dans les travaux présentés à APSEC 2012 [CCJM12b], ainsi qu'à SIMF 2013 [CC13].

8.1 Le sous ensemble *PLiMoS specification* pour une spécialisation

Le langage de modélisation PLiMoS$_{specification}$ (Ps) est le sous-ensemble du langage PLiMoS relatif à la définition de *relationships* entre éléments de modèles, et est défini par son domaine syntaxique abstrait $\mathcal{L}_{Ps}$, son domaine sémantique $\mathcal{S}_{Ps}$ et une fonction sémantique $\mathcal{M}_{Ps} : \mathcal{L}_{Ps} \rightarrow \mathcal{S}_{Ps}$ $(\llbracket \cdot \rrbracket_{Ps})$.

Définition 17. *Domaine syntaxique* $\mathcal{L}_{Ps}$
Un modèle de spécification $ms \in \mathcal{L}_{Ps}$ *est un 3-uplet* $\langle SBR, SI, \mathcal{M}_{SBR \rightarrow SI} \rangle$ *tel que :*

- SBR *est l'ensemble des structures des* relationships *de base, constructions sous-jacentes à la représentation créée ;*
- SI *représente l'ensemble des interprétations sémantiques possibles des* relationships *;*
- $\mathcal{M}_{SBR \rightarrow SI} \subseteq SBR \times SI$ *est l'ensemble des relations d'association entre la structure des* relationships *(BR) et les interprétations sémantiques réalisables (SP).*

En outre, chaque $ms \in \mathcal{L}_{Ps}$ *doit satisfaire les règles de bonne formation suivantes:*

- *Chaque structure de* relationship *de base respecte les règles de cohérence établies dans la section 8.2 ;*
- *Chaque interprétation sémantique respecte les règles de cohérence établies dans la section 8.3 ;*
- *Toute structure de relation est liée à au moins une interprétation sémantique définissant son rôle dans le processus :* $\forall si \in SI \exists sr \in SBR | sr \xrightarrow[\mathcal{M}_{SBR \rightarrow SI}]{} si$.

L'ensemble des combinaisons possibles des spécifications de *relationships* d'un espace de modélisation donné est appelé *domaine sémantique*, et est défini comme suit :

Définition 18. *Domaine sémantique* $\mathcal{S}_{Ps}$
$\mathcal{S}_{Ps} \triangleq \mathcal{P}(\langle SBR_i, SI_j \rangle_{\mathcal{M}})$, *avec* $SBR_i \in SBR$ *et* $SI_j \in SI$ *quelque soit* $i, j \in \mathbb{N}^*$ *, indique que tout modèle syntaxiquement correct doit être interprété tel un modèle de spécification des* relationships *d'une* ligne de produits.

L'opération liant le domaine syntaxique et les modèles valides correspondants est réalisée par la fonction sémantique.

Définition 19. *Fonction sémantique* $\llbracket ms \rrbracket_{Ps}$
$\mathcal{M} : \mathcal{L}_{Ps} \rightarrow \mathcal{S}_{Ps}$ *où* $\mathcal{M}(ms)$ *est l'ensemble des ensembles des paires valides* $p \in \mathcal{P}(\mathcal{M}_{SBR \rightarrow SI})$.
Chaque $p \in \mathcal{M}(ms)$ *est telle que:*

- *chaque extrémité est valide ;*
- *chaque structure de relation est reliée par une p ;*

8.2 Structure de la relation

L'intérêt est porté en premier lieu sur la structure de la *relationship*, son canevas, qui donne sa stabilité et sa cohésion, mais ne découvre pas à proprement parler son sens.

Une structure de *relationships* basiques $sbr \in SBR$, définissant l'ossature d'une relation, est exprimée par le 12-uplet $\langle A, BR, Nat, RE, CardSrcMin, CardSrcMax, CardTgtMin, CardTgtMax, Lhs, Rhs, EAR, MA \rangle$ tel que :

- A représente l'ensemble abstrait des artefacts de modélisation (A), reliés par les structures relationnelles ;
- $BR \subset A$ est l'ensemble des relations basiques dyadiques permettant la définition d'un maillage structurel *(BasicRelationship)* - BR ;

- $Nat : BR \rightarrow ARN$, est la fonction associant une propriété dérivée à chaque relation de base, avec ARN (*ArtifactRelatedNature*) défini tel que {*homogeneous,heterogeneous*} ;
- RE est l'ensemble des extrémités *(RelationshipEnd)* - RE des *relationship* de base ;
- $CardSrcMin : RE \rightarrow (\mathbb{N} \cup \{-1\})$ est la fonction donnant la cardinalité minimale côté source (-1 représentant *) ;
- $CardSrcMax : RE \rightarrow (\mathbb{N} \cup \{-1\})$ est la fonction donnant la cardinalité maximale côté source (-1 représentant *), avec $CardSrcMax > CardSrcMin$;
- $CardTgtMin : RE \rightarrow (\mathbb{N} \cup \{-1\})$ est la fonction donnant la cardinalité minimale côté cible (-1 représentant *), $CardTgtMin \neq \emptyset$;
- $CardTgtMax : RE \rightarrow (\mathbb{N} \cup \{-1\})$ est la fonction donnant la cardinalité maximale côté cible (-1 représentant *), avec $CardTgtMax > CardTgtMin$ et $CardTgtMax \neq \emptyset$;
- $Lhs : BR \rightarrow RE$ est la fonction d'association d'une extrémité gauche à une *relationships* ;
- $Rhs : BR \rightarrow RE$ est la fonction d'association d'une extrémité droite à une *relationships* ;
- $EAR \subset RE \times \mathcal{P}(A)$ est l'ensemble non vide des associations entre une extrémité et des artefacts de modélisation ($EAR \neq \emptyset$) ;
- $MA \subset A$ est l'ensemble des artefacts de type entité de modélisation existante, autre qu'une relation basique ($BR \cap MA = \emptyset$).

En outre, chaque $sbr \in SBR$ doit satisfaire les règles de bonne formation suivantes :

- Une extrémité de relation $re \in RE$ ne lie que des artefacts de même nature (soit des artefacts de modélisation externe soit des relations) :
  $\forall re \in RE \nexists a_1, a_2 \in A | a_1 \in MA \wedge a_2 \in BR$;
- Une relation basique possède obligatoirement deux extrémités :
  $\forall br \in BR \exists re_1, re_2 \in RE | Lhs(br) \rightarrow re_1 \wedge Rhs(br) \rightarrow re_2$;
- Chaque relation basique $br \in BR$ possède une nature associée $Nat(br) \neq \emptyset$, cette dernière est dérivée ;
- Chaque relation est unique :
  $\forall br_1 \in BR \nexists br_2 \in BR | Lhs(br_1) = Lhs(br_2) \wedge Rhs(br_1) = Rhs(br_2) \cdot br_1 \neq br_2$;
- Une relation ne s'autoréférence pas :
  $\forall br \in BR, a \in A, re \in RE \nexists < a, re > | Lhs(br) \rightarrow re = a \vee Rhs(br) \rightarrow re = a$.

La figure 8.1 propose un extrait de la représentation du métamodèle PLiMoS$_{definition}$ en ECore, relatif aux *relationships* basiques.

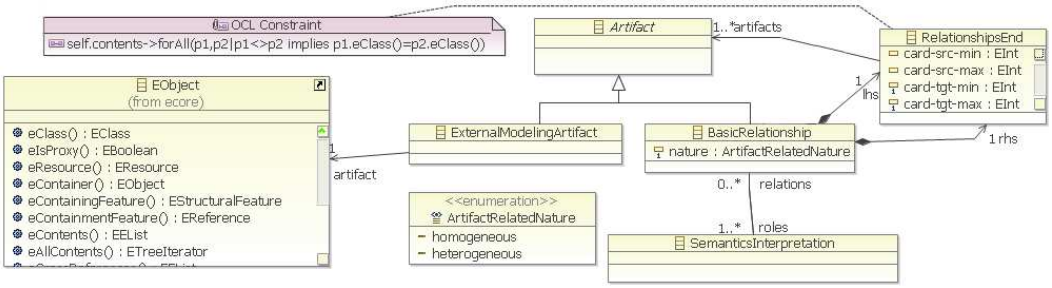


FIGURE 8.1: PLiMoS$_{definition}$: *relationships* basiques

La relation d'exclusivité entre des "types" d'artefacts liés par une extrémité de relations est vérifiée par la contrainte OCL exprimée par le listing 8.1, celle de l'autoréférencement prohibée par le listing 8.2.

```
context : RelationshipEnd
inv : self.oclContents->forAll(p1,p2|p1<>p2 implies p1.eClass=p2.eClass)
```

Listing 8.1: Contrainte OCL - même type d'artefacts connectés

```
context : RelationshipEnd
inv : not (self.lhs.artifacts->includes(self) or
        self.rhs.artifacts->includes(self))
```

Listing 8.2: Contrainte OCL - auto-référencement référentiel interdit

De même, l'implémentation ECore permet d'associer une expression OCL à un corps de dérivation d'un attribut, la détermination de la nature est réalisée par la fonction OCL du listing 8.3 (syntaxe *OCL in ECore*).

```
unsettable volatile }
        { derivation:
        if self.lhs.artifacts.oclType()->size() = 1
        and self.rhs.artifacts.oclType()->size() = 1
        and self.lhs.artifacts.oclType() = self.rhs.artifacts.oclType()
        then ArtifactRelatedNature::homogeneous
        else ArtifactRelatedNature::heterogeneous
        endif; }
```

Listing 8.3: Expression OCL - dérivation de la nature d'une relation

Un même canevas relationnel peut avoir un sens, par son interprétation sémantique associée, dans plusieurs de ces usages.

8.3 Interprétation sémantique des relations

Tout canevas relationnel structurel est associé à une ou plusieurs interprétations sémantiques. Les *relationships* ainsi définies sont caractérisées par une portée et une complexité, ainsi que d'autres propriétés listées dans les sections suivantes.

Les *relationships* trouvent leur utilité à travers plusieurs usages, dont :

- la modélisation, par ex. avec une décomposition et une organisation pour des modèles statiques, et une explicitation de séquencement d'événements pour des modèles dynamiques ;
- la vérification de cohérence, comme la cohérence logique des modèles de variabilité, la détection des conflits dans l'espace des *core assets* ;
- la composition, de modèles de variabilité par fusion ou de *core assets* (composition comportementale p. ex.) ;
- la traçabilité, pour retrouver une décision de conception, vérifier la prise en compte d'une exigence ;
- l'évolution, avec l'analyse d'impact de modifications ;
- la transformation de modèles, notamment pour la phase d'enrichissement lors de la réduction d'un niveau d'abstraction ;
- la dérivation de produits, et les étapes de vérification de la configuration et de construction du produit.

Les relations et leur interprétation sémantique associée ne possèdent pas de notion d'ordre, car il n'y a pas lieu de distinguer un quelconque ordre d'instanciation de ces dernières. Un couple

association interprétation est également unique, de part l'objectif sous-jacent de sa réalisation ; en effet, plusieurs relations de même type d'interprétation définissent une même activité, mais l'association structurelle des éléments n'a de sens que dans l'unicité d'une structure.

La présente section propose une formalisation des interprétations sémantiques avec une présentation du métamodèle en découlant avant de décrire plus en détail dans les sections suivantes les différentes propriétés.

8.3.1 Formalisation des interprétations sémantiques

Une interprétation sémantique des *relationships* $isr \in SBR$, est exprimée par le 26-uplet $\langle SI, Al, Cr, Crit, Desc, IsDef, Nat, Temporality, Impact, Merged, KH, ER, E, O, OR, DE, InfFrom, HSS, LHSR, RHSR, REI, REIR, QA, QAR, PVal, PValR\rangle$, tel que :

- SI est l'ensemble des interprétations sémantiques possibles d'une structure relationnelle de base ;
- $Al : SI \rightarrow MLO$ est une fonction dans l'énumération *ModelingLevelOrientation*, $MLO = \{horizontal, vertical, both\}$, renseignant sur l'attachement de la relation vis-à-vis du processus de modélisation ;
- $Cr : SI \rightarrow CT$ est une fonction dans l'énumération *ConcernType*, qualifiant la relation d'intra ou inter préoccupation ($\{inter, intra\}$) ;
- $Crit : SI \rightarrow CritL$ est une fonction dans l'énumération *CriticityLevel*, qualifiant la relation par un niveau de criticité pour un processus ou une activité donnée ($\{high, medium, low\}$), par ex. lors de la configuration de produit, certains éléments peuvent être considérés comme nécessaires, d'autres pouvant être souhaitables à divers degrés (relativement critique - *warning* - ou donné à titre informatif) ;
- $Desc : SI \rightarrow TD$ est une fonction dans l'ensemble des descriptions textuelles (TD) possibles de la relation ;
- $IsDef : SI \rightarrow \{True, False\}$ est une fonction dans l'ensemble des booléens permettant d'indiquer la qualité de relation par défaut dans le cas de relations abstraites et dérivées ;
- $Nat : SI \rightarrow NK$ est une fonction dans l'énumération *NatureKind*, $NK = \{abstract, concrete, derived\}$, relative à la nature de la relation. Une relation abstraite est non instanciable, une relation dérivée est non sérialisable ;
- $Temporality : SI \rightarrow TU$ est une fonction dans l'énumération *TimeUsage*, $TU = \{design, run, over\}$, relative à la temporalité de la relation ;
- $Impact : SI \rightarrow RImp$ est une fonction dans l'énumération *RelationImpact*, qualifiant l'impact de la relation ($\{induction, activation, modification, none\}$) ;
- $Merged : SI \rightarrow \{True, False\}$, à faux par défaut, correspond à une alternative de génération du langage de spécialisation, maillage et sémantique imbriqués ou non ;
- $KH : SI \rightarrow \{True, False\}$, à vrai par défaut, correspond à une alternative de génération du langage de spécialisation, génération de classes abstraites pour qualifier la sémantique par héritage ;
- $ER \subseteq SI \times \mathcal{P}(SI)$ est l'ensemble des associations entre une interprétation sémantique et celles qu'elle étend (*extends*) ;
- E est l'ensemble abstrait des expressions (E), ayant pour propriétés un langage (avec dans la pratique une prise en considération des langages suivants $Lang : E \rightarrow \{ocl, acceleo, kermeta, java, other\}$) et un corps exprimé en ce langage ($Body : E \rightarrow DB$, avec DB la description du corps de manière textuelle) ;
- O est l'ensemble des opérations ($O \subseteq E$), ayant pour propriété une signature $Sign : E \rightarrow IS$, avec IS représentant l'identification de la signature de manière textuelle ;

- $OR \subseteq SI \times \mathcal{P}(O)$ est l'ensemble des associations entre une interprétation sémantique et des opérations ;
- DE est l'ensemble des expressions de dérivation (*DerivationExpression* - $DE \subseteq E$) des interprétations sémantiques ;
- $InfFrom : SI \rightarrow DE$ est une fonction dans l'ensemble DE renseignant sur la spécification de la dérivation d'une relation ;
- HSS est l'ensemble des déclarations concernant les éléments gauche et droite des relations (*HandSideStatement* - $HSS \subseteq E$, et $O \cap DE \cap HSS = \emptyset$), ayant pour propriété un identifiant $Ident : HSS \rightarrow IdS$, avec IdS représentant la désignation de l'identifiant de manière textuelle ;
- $LHSR \subseteq SI \times \mathcal{P}(HSS)$ est l'ensemble des associations entre une interprétation sémantique et une ou plusieurs déclarations de valeur gauche ;
- $RHSR \subseteq SI \times \mathcal{P}(HSS)$ est l'ensemble des associations entre une interprétation sémantique et une ou plusieurs déclarations de valeur droite, avec $LHSR \cap RHSR = \emptyset$;
- REI est l'ensemble des interprétations des extrémités *(RelationshipEnd)*. Ces derniers ont pour propriétés : un identifiant $Idti : REI \rightarrow IdtiS$, avec $IdtiS$ représentant la désignation de l'identifiant de manière textuelle ; un nom de source $SrcN : REI \rightarrow IdtsrcS$, avec $IdtsrcS$ représentant la désignation de la source de manière textuelle ; un nom de cible $TgtN : REI \rightarrow IdttgtS$, avec $IdttgtS$ représentant la désignation de la source de manière textuelle ($IdttgtS \neq \emptyset$) ; une option de réification $Reif : REI \rightarrow \{True, False\}$, à faux par défaut ;
- $REIR \subseteq SI \times \mathcal{P}(REI)$ est l'ensemble des associations entre une interprétation sémantique et deux interprétations des extrémités ($Card(REIR) = 2$) ;
- QA est l'ensemble des attributs de qualité (*QualityAttribute*). Ces derniers ont pour propriétés : un identifiant $Idt : QA \rightarrow IdtS$, avec $IdtS$ représentant la désignation de l'identifiant de manière textuelle ; une description $Desct : QA \rightarrow TDesc$ est une fonction dans l'ensemble des descriptions textuelles ($TDesc$) possibles de l'attribut ; une propriété quantitative $Many : QA \rightarrow \{True, False\}$, à faux par défaut ; une option de réification $Reified : QA \rightarrow \{True, False\}$, à faux par défaut ;
- $QAR \subseteq SI \times \mathcal{P}(QA)$ est l'ensemble des associations entre une interprétation sémantique et des attributs de qualité ;
- $PVal$ est l'ensemble des valeurs possibles d'un attribut de qualité ($PVal \rightarrow PValText$ avec $PValText$ une description textuelle de la valeur) ;
- $PValR \subseteq QA \times \mathcal{P}(PVal)$ est l'ensemble des associations entre un attribut de qualité et une ou plusieurs valeurs, avec $PValR \neq \emptyset$.

Sur la base d'une interprétation sémantique des *relationships* $isr \in SBR$ est définie une interprétation spécifique aux *lignes de produits*, $plisr \in PLSBR$ ($PLSBR \subseteq SBR$), exprimée par le 3-uplet $\langle PLSI, Prop, PropCoord \rangle$, définit tel que :

- $PLSI \subseteq SI$ est l'ensemble des interprétations sémantiques possibles d'une structure relationnelle de base d'un *ligne de produits* ;
- $Prop : PLSI \rightarrow SP$ est une fonction dans l'ensemble des propriétés sémantiques prédéfinies pour le domaine des *lignes de produits* (description détaillée en section 8.4) ;
- $PropCoord : PLSI \rightarrow CoordT$ est une fonction dans l'énumération *CoordinationType*, qualifiant la relation de coordination ($\{implication, coimplication, exclusion, none\}$) ;

En outre les interprétations sémantiques doivent respecter certaines dépendances et contraintes entre les propriétés, des précisions sont apportées dans la section suivante (8.4). Par ailleurs, la figure 8.2 propose un extrait de représentation du métamodèle PLiMoS$_{definition}$ relatif aux interprétations sémantiques.

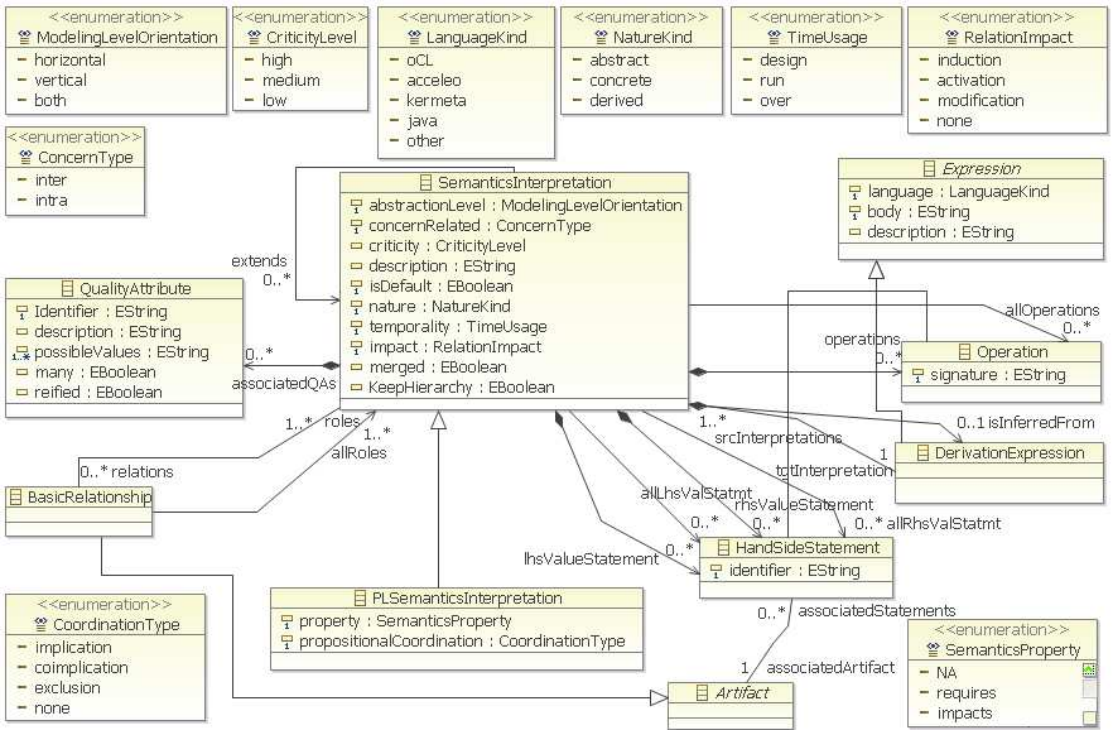


FIGURE 8.2: PLiMoS$_{definition}$: interprétations sémantiques

8.3.2 Portée des *relationships*

Le rayon d'action détermine la portée de la définition des *relationships* (définies par le canevas structurel de *BasicRelationships* et une interprétation sémantique associée), selon :

- le rapport aux artefacts : qui détermine le caractère homogène ou hétérogène de la relation (cette propriété est relative au canevas de la relation) ;
- le niveau de granularité : qui est défini, dans notre contexte, comme inter-LdP, inter-modèles ou intra-modèle (dans une hiérarchie de modélisation, cette catégorie peut être raffinée, par ex. entre composants, sous-composants et objets) ; dans un contexte plus large, ce niveau peut englober des relations écosystèmes (cette propriété dépend également du canevas et des types d'artefacts mis en relations) ;
- la relation à l'abstraction : qui établit l'appartenance à un même niveau d'abstraction - *relationships* horizontale - ou transverse à deux niveaux - *relationships* verticale ;
- l'attachement au niveau de modélisation : qui replace la relation dans son contexte de métamodélisation. Les relations sont déterminées dans une approche à trois niveaux entre modèles, entre métamodèles, voire entre méta-métamodèles (cadre de multiples espaces technologiques), et plus dans une approche multi-niveaux (cette propriété n'est pas réifiée dans l'approche, qui se focalise sur le niveau modèle plus particulièrement) ;
- la temporalité : qui assigne la pertinence de la relation à un moment donné, lors de la conception (*design*), de la compilation, de l'exécution (*run*), au cours du temps (évolution - *over time*) ;
- la préoccupation ciblée : qui situe la relation dans son usage de description d'une préoccupation (*intra-concern*), ou de lien transverse à plusieurs préoccupations (*inter-concern*).

8.3.3 Complexité des *relationships*

Les *relationships* présentent divers niveaux de complexité de mise en relation :

- le référencement direct : il s'agit d'établir un lien point à point entre deux éléments, une variation permettant de choisir la navigabilité (unidirectionnalité ou bidirectionnalité), pondérant ainsi l'importance des participants ;
- le référencement matriciel : la mise en relation se complexifie, et lie un ou plusieurs ensembles d'éléments avec d'autres ; une catégorie de relations qui s'exprime classiquement par une notation mathématique d'uplets, et qui permet une relation *1 vers n*, *n vers n*, et *n vers 1* ; la navigabilité est également spécifiable ;
- le référencement logique : l'expression de logique de premier ordre s'exprime au travers d'une combinaison d'éléments, sous réserve de la validation d'une garde évaluant la pertinence de la sélection ;
- le référencement et actuation : cette mise en relation se différencie de la précédente par la conséquence de son franchissement qui intègre la capacité d'actuation d'éléments, de modification de valeurs, de calculs ou de transformations de modèles.

Les référencements directs et matriciels s'entendent d'eux-mêmes, le dernier permettant déjà une conjonction de plusieurs éléments. Les deux dernières catégories font appel aux éléments *HandSideStatement* et *Operation* décrits plus en détail par la suite.

8.3.4 Symétrie, dissymétrie, transitivité et navigabilité

La définition d'une structure de relation d'une modélisation par les *Basic Relationships* propose une symétrie par défaut, avec l'identification de chacune des extrémités par la séparation d'une partie gauche (*LHS*) et droite (*RHS*), cf. section 8.2. La propriété sémantique *propositional coordination* induit une logique de coordination impactant cette symétrie de la relation, de même que sa transitivité et sa navigabilité.

Les différentes caractéristiques sont décrites ci-après :

- la symétrie de la relation : la propriété *propositional coordination* aux valeurs co-implication ou exclusion conserve la symétrie initiale de la relation, à l'inverse l'implication simple la rend dissymétrique (la valeur *none* ne précise rien, ne génère pas de méthodes de navigation et est donc à réserver pour des cas bien identifiés) ;
- la navigabilité : une relation symétrique est navigable dans les deux sens (vers la gauche ou la droite) alors qu'une relation dissymétrique permet une navigation de la source (anciennement identifiée par gauche) vers la cible (anciennement identifiée par droite). Des opérateurs de navigabilité sont générés dans ce sens, cf. chapitre 9 (remarque : pour *none* aucun opérateur n'est généré automatiquement) ;
- la transitivité : une relation est transitive si elle est considérée comme d'implication ou de co-implication.

Le tableau 8.1 résume l'état du triplé - symétrie, navigabilité et transitivité - en fonction de la valeur de sélection de la propriété *propositional coordination*.

Tableau 8.1: Récapitulatif - symétrie, navigabilité et transitivité - en fonction de la coordination.

Propositional Coordination	symétrie	navigation	transitivité
implication	dissymétrie	src $\rightarrow$ tgt	oui
co-implication	symétrie	lhs $\leftrightarrow$ rhs	oui
exclusion	symétrie	lhs $\leftrightarrow$ rhs	non
none	symétrie	aucune *a priori*	non

8.3.5 Expressivité de coordination des *relationships*

La structure relationnelle permet de mettre en rapport des éléments à gauche avec des éléments à droite (opérateur *ET* logique). La négation d'un ou plusieurs éléments des extrémités permet l'application de l'opérateur *NOT* sur les entités ciblées.

Une structure de base exprime par défaut une conjonction copulative dénotant la simultanéité (et), qui peut se compléter par une "négation" (pour exprimer un ni). Par ailleurs, la disjonction requiert l'apposition de deux interprétations sémantiques sur une même structure, la différentiation de sélection s'effectuant par le biais des gardes et de leurs valeurs. En outre, une coordination consécutive (donc) est associée à une proposition d'implication, de co-implication, ou d'exclusion. À l'inverse, une coordination causale (car) n'est pas associée à une propriété d'implication par défaut (il est nécessaire d'ajouter des opérations utilisateur pour modifier la navigabilité).

Pour faire un lien avec la section précédente, il est possible d'exprimer la sémantique de coordination des relations symétriques et dissymétriques. Tout d'abord, la dissymétrie exprime que si *gauche* est vérifié (condition de sélection, de valeurs, etc.), alors *droit* devient vrai (et permet une implication, une modification, etc.), et la contraposée l'est également.

La symétrie exprime une symétrie de sens et pose une relation logique identique, soit : si *LHS* (respectivement *RHS*) alors *RHS* (respectivement *LHS*).

8.3.6 Relations abstraites et dérivées

L'habileté à inférer des relations implicites entre modèles apporte une aide pour renforcer la sémantique de cohésion entre les ensembles de modèles dépendants. L'approche vise à améliorer l'évolution de l'infrastructure de la *ligne de produits* en reliant ses points de variation aux arguments rationnels de leur existence et, par là, à soutenir la dérivation de nouveaux membres de la *ligne de produits*.

Des opérations suivant une certaine sémantique de haut niveau peuvent apparaître, associées à des relations abstraites. Cependant, certaines conditions sont nécessaires à l'extension d'une abstraction ; tout d'abord, le canevas structurel doit être identique, de même que le niveau d'abstraction ainsi que la préoccupation. La coordination suit les règles énoncées ci-après : une implication peut se spécialiser en une implication ou une co-implication ; une co-implication peut se raffiner uniquement en une co-implication ; et pour terminer, une exclusion reste une exclusion et une none peut prendre toute valeur. Enfin, dans le cas d'une définition d'opérations ou d'attributs de qualité, une re-déclaration de signature à l'identique correspond à une redéfinition de ces derniers (*overriding*).

Les relations abstraites ne sont pas "instanciables" dans le langage, elles permettent une hiérarchisation de ce dernier et, par exemple, des facilités d'interopérabilité avec les autres langages de variabilité (cf. section 9.1.1).

Les relations dérivées sont considérées comme volatiles et "*transient*", à savoir qu'elles sont calculées (le corps étant défini par un langage d'action existant) et ne sont en aucun cas sérialisées. Elles sont exprimées par un langage prédéfinit, par l'instanciation d'une entité *DerivationExpression*. Les relations *srcInterpretations* sont également dérivées et volatiles, calculées à partir des éléments de la *DerivationExpression*.

8.3.7 Impact des relations

L'impact d'une relation se qualifie en termes d'interaction entre des éléments. Les interactions prises en considération sont dans ce mode de description toutes explicites et directes, c.-à-d. que l'élément déclencheur interagit avec l'élément déclenché, et que le déclenché dépend directement du déclencheur.

Tout d'abord, (i) l'induction *(induction)* correspond à une sélection ou désélection d'un élément (dans le cadre d'une description comportementale il s'agit d'une invocation). Ensuite, (ii) l'activation *(activation)* concorde avec une réalité davantage dynamique que la précédente, correspondant à l'activation d'un élément, d'un composant, à l'invocation d'un service, qui présuppose la sélection, et donc la disponibilité de l'élément déclenché en question. Enfin, alors que les deux premiers types d'impact ne présentent pas un caractère altérant, (iii) la modification *(modification)* décrit une modification, production, suppression d'une ressource ou d'un de ces semblables. Dans le cas d'un changement de comportement, il est important de s'intéresser à l'impact de ce dernier sur les autres (vérifier les interactions possibles) afin de ne pas altérer le système.

Les interactions de niveau métier reflètent la capacité d'un système à se modifier lui même. Dans le cadre des *lignes de produits* et de la modélisation de la variabilité, la sélection d'un variant peut entrainer une modification sur les contraintes du modèle de variabilité : restriction ou élargissement de cardinalité par exemple, ou dans le cas d'un modèle de variabilité ayant une capacité d'expression d'attributs de variant, modification de la valeur de ces derniers.

8.3.8 Autres propriétés

Quelques autres propriétés sont présentées ci-après :

- Attributs de qualité : *QualityAttribute* permet l'association de la *relationship* avec des attributs de qualité définissant une énumération possible de valeurs. Ils peuvent être multiples et réifié lors de la génération du langage spécifique ;
- Gardes et actions : *HandSideStatement* permet, suivant la position dans la relation d'exprimer une garde ou une action sur des artefacts de la structure (dans un langage prédéfini). Une garde est un prédicat, c.-à-d. un morphisme de l'ensemble des éléments de modèle considérés par l'expression vers l'ensemble des valeurs de vérité. Une action permet une modification des éléments ciblés par l'expression, par ex. une màj de valeurs. Les relations *allLhsValues* et *allRhsValues* sont dérivées, de même que les relations *associatedArtefacts* ;
- Opérations : *Operation* permet d'associer diverses opérations définies par l'utilisateur à une interprétation sémantique relationnelle.

Indépendamment des opérations définies par l'utilisateur, les gardes et actions sont associables à la relation. La structure relationnelle se voit donc attribuer potentiellement (par l'entité *SemanticInterpretation*) une valeur gauche (*lhsValueStatement*) et une valeur droite (*rhsValueStatement*), toutes deux de type *HandSideStatement* (défini par un identifiant une description, un langage et un corps). Dans le cas d'une relation dissymétrique (c.-à-d. d'implication), si présence il y a d'une valeur gauche elle est interprétée telle une garde, alors que la valeur droite est interprétée telle une action à réaliser. Dans le cas d'une symétrie, bidirectionnelle, les *HandSideStatements* sont par intermittence des gardes et des actions.

8.4 Propriétés sémantiques des *relationships* pour les LdPs

Certaines propriétés sémantiques spécifiques au domaine des *lignes de produits* dirigées par les modèles sont intégrées au langage PLiMoS. Cette section propose une description de ces propriétés et fournit une description des contraintes liées à la sélection de chaque propriété sémantique, par exemple une relation *require* induit une coordination de type *implication*.

Les propriétés sémantiques sont décrites suivant un classement usuel commun d'application pour des relations. Cette délimitation n'est pas et ne doit pas être une limitation dans la pratique, notamment du fait de la multitude de choix d'abstraction des modèles de variabilités.

8.4.1 Dans l'espace de variabilité

Les propriétés couvrent les besoins classiques d'expression relationnelle des modèles de variabilité.

Relations structurelles hiérarchiques

- Généralisation (spécialisation) : couplage non symétrique sémantiquement, qui permet de créer un genre (hyperonyme) à partir d'espèces (hyponyme), de créer des super types à partir de sous-types (*parent - child*).
- Classification : (*is a*) permet de créer des types à partir d'instances.
- Agrégation : association non symétrique, qui exprime un couplage fort et une relation de subordination de type "ensemble / élément". Une paire d'éléments peut être reliée par une relation de type *has a*, marquant une décomposition structurelle. Les cycles de vies de l'agrégat et de ses éléments agrégés peuvent être indépendants.
- Composition (décomposition): soit *part-of*, *element-of*, *decomposed-in*, représente une relation plus soutenue d'agrégation entre éléments (agrégation par valeur), où les cycles de vies des éléments (les composants) et de l'agrégat (le composite) sont liés : si l'agrégat est détruit (ou copié), ses composants le sont aussi.

Relations de configuration et variations sémantiques

Les relations de configuration expriment des conformances causales. Elles sont de trois grandes familles qui peuvent être spécialisées (cf. section suivante), à savoir :

- *requires* : un élément e_1 en nécessite un autre e_2 pour fonctionner. En termes d'exigences de fonctionnement, r_1 est satisfaite, seulement quand r_2 est atteinte.
- *co-implies* : les éléments e_1 et e_2 sont indissociables pour fonctionner.
- *excludes* : deux éléments ne peuvent fonctionner de concert.

Divers usages de configuration

Plusieurs variations sémantiques étendent les trois types de configuration, par exemple :

- *implication* : *impacts*, *implies*, *includes*, *extends*, *requires*, *uses*, *provided-by*, *contributes-to*, *influences*, *depends-on*,
- *co-implication* : *co-implies*, *realized-by*,
- *exclusion* : *excludes*, *incompatible with*, *mutex-with*, *conflicts* et *inconsistent-with* (représentation d'incohérence, de type dérivée).

Certaines sémantiques insistent sur le fait qu'un élément ne peut remplir ses obligations *ssi* un autre élément est présent, à l'exemple de *requires*, *uses* et *depends-on* ; d'autres s'attachent à décrire l'interaction d'un élément avec un autre, par ex. *impacts*, et *extends*.

Ci-après sont listées quelques variantes d'implication :

- *hints* : un variant a une influence positive sur un autre ; exemple l'espace mémoire faible a un impact sur la taille de la carte.
- *hinders* : un variant a une influence négative sur un autre ; exemple : une fréquence de processeur élevé entraîne une augmentation de moyens de refroidissements.
- *recommends* : une forme moins critique de *requires*, elle peut par ex. servir à exprimer une valeur par défaut.
- *discourages* : une forme allégée de *excludes* de criticité moyenne.
- *interacts* : il y a interaction entre deux variants si l'un modifie le comportement de l'autre. Généralement ces variants sont reliés à des modèles d'exécution.
- *impacts* : la sélection d'un élément a un impact sur un autre (sans plus de détails sur le type d'impact, ni sur son caractère positif ou négatif).

Abstraction et verticalité

Par ailleurs, certaines relations sont caractérisées par leur verticalité. Les relations verticales sont relatives à une précision de détails et de granularité, et également au franchissement d'une "frontière" marquée entre des niveaux d'abstractions. L'abstraction, outre son caractère par *schématisation*, caractérisant la notion même de modèle (qui ne fait pas l'objet de l'étude), permet une définition de niveaux de modélisation. Des relations d'abstraction / raffinement parsèment la description de l'espace de modélisation.

- *satisfies* : un élément répond aux attentes, aux besoins et aux désirs d'un autre.
- *realized-by* / *provided-by* : décrit la notion de réalisation d'un élément plus abstrait par un autre.
- *implements* : décrit la notion clé entre une implémentation, un élément concret, et sa spécification, un élément abstrait. De manière informelle, la relation peut être exprimée de la manière suivante : un élément x implémente un élément plus abstrait y *ssi* x exhibe la sémantique spécifiée par y, en terme de comportement, de structure, etc.
- *refines* : (raffinement) le raffinement peut être de diverse nature, intra et extra modèles soit hétérogène ou homogène. Cette relation reste générique et les relations de spécialisation, de réalisation, de satisfaction, et d'implémentation par exemple apportent une sémantique adaptée à chaque situation. De manière globale, le raffinement propose une description davantage détaillée (de l'implémentation, des services) d'un élément.
- *allocated-to* : un élément est alloué à une certaine ressource physique.

Concernant la décomposition structurelle inter-modèle et homogène, des relations de généralisation / spécialisation, classification, agrégation et décomposition font office de relations de raffinement.

Similarité et chevauchement sémantique

Alors que les modèles de variabilité trouvent leur fondement dans l'expression des différences, certaines relations se dédient à l'expression de similarités. Deux éléments substituables peuvent différer dans leurs caractéristiques non-fonctionnelles, telles que le temps d'exécution, l'espace mémoire requis, le coût, la licence, les restrictions légales d'usage, le niveau de confiance de conformité, etc.

À un niveau de modélisation donné, sur une modélisation inter-modèle (même "type" de modèles), une équivalence entre deux éléments traduit que ces deux éléments font référence à un élément commun du monde réel et représentent donc un concept identique. Cette relation est qualifiée de bissimilaire mathématiquement parlant.

Ces relations permettent une définition du chevauchement entre des modèles *(overriding)* que se soit à un même niveau et type de modélisation avec une finalité de fusion *(merge)* de modèles, ou entre deux types de modèles pour de la vérification de cohérence (entre modèle de classe et diagrammes à états par exemple). En effet, f_1 équivalent à f_2 signifie $[\![f_1]\!] = [\![f_2]\!]$, même sémantique. Y apparaissent les appellations *same*, *equivalent*, *identity*.

Une relation de correspondance sémantique entre éléments de modèles ("*overlapping elements*") est de grande importance pour raisonner sur les modèles, de même que pour les intégrer et les réconcilier.

Relations et comportements

Des relations telles celles de hiérarchisation ou de configuration présentées précédemment sont qualifiées de statiques ; à l'inverse d'autres peuvent marquer une dynamicité par la description d'un comportement désiré ou une modélisation à l'exécution (*runtime*).

Les dépendances d'activation sont au nombre de cinq :

- *excluded activation* : un élément exclut le fonctionnement d'un autre ;
- *required activation* : (*subordinate activation*) un élément peut être actif, *ssi* un autre est actif dans la même fenêtre temporelle ;
- *concurrent activation* : (*collateral activation*) deux éléments sont actifs de manière concurrentielle. L'activation est simultanée, et la dépendance est donc bidirectionnelle. La désignation *collateral* en est un synonyme.
- *sequential activation* : un élément est actif à la suite d'un autre. L'activation de deux éléments est séquentielle, l'un juste après l'autre. Ce type de dépendance peut représenter des flux de contrôle ou des opérations de flux de données. La désignation *serial* en est un synonyme.
- *synergetic activation* : deux éléments ou plus doivent se synchroniser sporadiquement durant leur période d'activation. Ces relations représentent la présence de relations séquentielles dans un environnement de concurrence.

Dédié à une préoccupation donnée

Certaines sémantiques s'orientent vers une préoccupation spécifique, en voici quelques exemples :

- *resource usage* : dans le cas de ressources disponibles de manière limitée dans le temps, cette sémantique marque une exclusion temporairement limitée en fonction de la préemption d'une resource donnée ;
- *environment induced* : un élément est lié à son environnement physique ou à un contexte modélisé à un certain niveau d'abstraction ;
- *provided by* : dans le cas de services, un élément est fourni par un autre, induisant une dépendance de type nécessité et séquencement ;
- *invokes*, *synchronizes with*, *excludes*, *preempts* : dans le cas de structures de processus et de procédures, utilisés à des fins d'amélioration des performances du système, elles permettent la traçabilité des flux de contrôles, le débogage, et permet d'améliorer la traçabilité, la maintenance, et la compréhension.

Récapitulatif des propriétés

Les propriétés sémantiques sont liées à d'autres attributs de définition de la relation. Le tableau 8.2 présente les contraintes associées à la sélection de chaque propriété sémantique de l'espace de variabilité. Dans la pratique, des contraintes OCL sont établies afin de vérifier le comportement souhaité. À l'exemple du listing 8.4 qui présente un invariant spécifiant que la propriété à *generalization*, implique automatiquement d'avoir un type *interconcern*, une coordination de type *implication*, un niveau de criticité *high*, et un impact de type *induction*.

```
context : PLSemanticsInterpretation
inv : property = SemanticsProperty::generalization
        implies concernRelated = ConcernType::inter
        and propositionalCoordination = CoordinationType::implication
        and criticity = CriticityLevel::high
        and impact = RelationImpact::induction
```

Listing 8.4: Contrainte OCL - sur la propriété sémantique *generalisation*

Tableau 8.2: Propriétés sémantiques prédéfinies - espace de variabilité.

Nom	*Propriétés associées*
generalization	intraconcern, implication, high criticity level, induction
classification (*is a*)	intraconcern, implication, high criticity level, induction
aggregation (*has a*)	horizontal modeling level, intraconcern, implication, high criticity level, induction
composition (*part-of*, *element-of*, *decomposed-in*)	horizontal modeling level, intraconcern, implication, high criticity level, induction
requires	horizontal modeling level, intraconcern, implication, high criticity level, induction
includes	horizontal modeling level, intraconcern, implication, high criticity level
extends	horizontal modeling level, intraconcern, implication, high criticity level
provided-by	horizontal modeling level, intraconcern, implication, high criticity level, induction
uses	intraconcern, implication, high criticity level, induction
contributes-to	intraconcern, implication, high criticity level, modification
influences	intraconcern, implication, high criticity level, modification
depends-on	intraconcern, implication, high criticity level
impacts	implication, high criticity level
implies	implication, high criticity level, induction
co-implies	horizontal modeling level, intraconcern, co-implication, high criticity level, induction
realized-by	vertical modeling level, intraconcern, co-implication, high criticity level, induction
excludes	horizontal modeling level, intraconcern, exclusion, high criticity level
incompatible-with	intraconcern, exclusion, high criticity level, induction
mutex-with	intraconcern, exclusion, high criticity level, induction
conflicts	intraconcern, exclusion, high criticity level
inconsistent-with	horizontal modeling level, intraconcern, exclusion, high criticity level, induction
hints	intraconcern, implication, medium criticity level
hinders	intraconcern, exclusion, medium criticity level
discourages	intraconcern, exclusion, medium criticity level
recommends	intraconcern, implication, low criticity level
interacts	intraconcern, co-implication, high criticity level, run
satisfies	vertical modeling level, interconcern, implication, high criticity level, design, induction
realized-by / provided-by	vertical modeling level, interconcern, co-implication, high criticity level, design, induction
implements	vertical modeling level, interconcern, implication, high criticity level, design, induction

à suivre...

Nom	*Propriétés associées*
suite...	*suite...*
refines	vertical modeling level, interconcern, implication, high criticity level, design, induction
allocated-to	vertical modeling level, interconcern, implication, high criticity level, design, induction
equivalent	horizontal modeling level, intraconcern, co-implication, high criticity level
excluded-activation	horizontal, intraconcern, exclusion, run time, activation, high level
required-activation	horizontal, intraconcern, implication, run time, activation, high level
concurrent-activation	horizontal, intraconcern, co-implication, run time, activation, high level
sequential-activation	horizontal, intraconcern, implication, run time, activation, high level
synergetic-activation	horizontal, intraconcern, co-implication, run time, activation, high level
resource-usage	horizontal, intraconcern, exclusion, high level
environment-induced	horizontal, interconcern, implication, induction, high level
provided by	
flow-invokes	horizontal, intraconcern, implication, activation
flow-synchronizes-with	horizontal, intraconcern, co-implication, activation
flow-excludes	horizontal, intraconcern, exclusion, activation
flow-preempts	horizontal, interconcern, implication, induction

8.4.2 Dans l'espace de modélisation des *core-assets*

Les relations du langage PLiMoS couvrent non seulement les relations de l'espace de variabilité, mais également celles de l'espace de modélisation des *core-assets*, considérant ce dernier comme défini suivant le formalisme UML 2.

Dans les diagrammes de *Use-cases*

Les diagrammes de *Uses-cases* mettent classiquement en relation des acteurs avec des cas d'utilisation par des interactions de nature spécifique décrites par une expression. Trois types de relations existent entre les cas d'utilisation : *includes* (dans ce type d'interaction le premier cas englobe l'autre et son issue dépend souvent de la résolution du second), *extends* (les extensions représentent des prolongements logiques de certaines tâches sous certaines conditions, c.-à-d. qu'un cas d'utilisation a étend un cas d'utilisation b lorsque le cas d'utilisation a peut être appelé au cours de l'exécution du cas d'utilisation b) et *generalization* (le cas d'utilisation a est une généralisation de b, si b est un cas particulier de a c.-à-d. lorsque a peut être substitué par b pour un cas précis).

Dans les diagrammes de classes

Les modèles de diagrammes de classes proposent les relations suivantes :

- *Héritage* : la relation d'héritage permet une spécialisation, une généralisation ou une classification (*inherits-from*) ;

- La relation de *réalisation* (*implements*) définit un respect du contrat spécifié par une interface ;
- L'*association*, l'*agrégation* ainsi que la *composition* sont également présentes, liées à des propriétés de multiplicité, afin de décrire la structure composite du modèle ;
- La relation entre la classe et les objets associés définit la notion d'instanciation avec *is instance of* ;
- Des relations de *dépendances* qui décrivent des relations limitées dans le temps entre des objets instances des classes décrites sont également présentes.

Dépendances dans la modélisation des composants

Les composants logiciels sont, par nature et définition, construits pour être autosuffisants, ceci n'exclut cependant pas qu'un composant ait des dépendances externes.

La relation *requires* exprime le besoin (i) d'un service fourni par d'autres composants, (ii) d'une fonctionnalité fournie par des ressources externes, telles que celles offertes par la couche d'intergiciel, ou le système d'exploitation. Une analyse des dépendances entre composants et couches supérieures met en avant des relations de réalisation, d'implémentation et de dépendance fonctionnelle. Par exemple, un composant peut nécessiter des services d'autres composants pour pouvoir implémenter ses responsabilités ($c1$ *implements* $c2$).

Une partie de ces informations de dépendance externe se trouve représentée dans l'interface (services "*provided*" ou "*required*"). Néanmoins cette source d'information est parcellaire, l'interface ne renseignant pas sur les conditions d'applicabilité du service, sur les objectifs sous-jacents, etc.

Un composant logiciel peut nécessiter un conteneur particulier ou une ressource matérielle pour s'exécuter. Ces relations peuvent s'exprimer soit dans un modèle de *features*, soit dans des modèles relationnels découplés du modèle de variabilité.

En outre, les composants logiciels ont également des dépendances internes :
- dépendances entre éléments du composant ;
- dépendances entre intrants et extrants (dans le cas de composants réactifs) ;

L'importance de ces composants est grande lorsqu'il s'agit de traiter l'évolution et la modification des composants.

Dépendances dans les modèles comportementaux

Les dépendances d'activation sont nombreuses : *multiple*, *exclusive*, *subordinate*, *concurrent*, *sequential*, *predecessor*, *successor*. Afin d'exprimer formellement la sémantique, cette dernière repose sur les quatre relations suivantes (considérons a et b, deux éléments de modèle de la modélisation comportementale ($a \neq b$)) :

- *finishToStart* : un élément b ne peut pas démarrer avant qu'un élément a n'ait terminé. Ce type de dépendance est une des plus, voir la plus, commune relation des diagrammes de flux, définissant l'idée même de séquence.
- *finishToFinish* : un élément b ne peut pas se terminer avant qu'un élément a ne se soit terminé.
- *startToStart* : un élément b ne peut pas démarrer avant qu'un élément a ne démarre. Cette dépendance, combinée à la précédente (*finishToFinish*) permet de spécifier un parallélisme entre plusieurs groupes d'éléments, l'embranchement initial (*fork*) étant marqué par des relations *startToStart*, et la jointure (*join*) par des relations *finishToFinish*.
- *startToFinish* : un élément b ne peut pas terminer avant qu'un élément a n'ait démarré. Ce type de dépendance trouve une application dans la modélisation d'ordonnancements *just-in-time*.

Remarque : ces relations sont génériques, celles décrites dans la section 8.4.1 sont spécifiques aux besoins et aux habitudes de la communauté des *lignes de produits*.

Dépendances dans les modèles au cours du temps

L'évolution durant le développement, la maintenance, est identifiable par une série de sémantiques de relation, et sont identifiées comme suit : *has-evolved-to*, ou *overrides*, un élément e1 a évolué et a été remplacé par un élément e2, les deux éléments ayant un rôle identique dans la modélisation. D'autres sémantiques permettent de prévoir et de mieux appréhender le processus d'évolution (*is evoling to*, *will evolve to*, *may evolve to*). Ces sémantiques autorisent à revoir le statut d'un élément en fonction de l'évolution des connaissances, et permettent d'éviter des incompréhensions et disfonctionnements.

Récapitulatif des propriétés

De même que dans l'espace de variabilité, les propriétés sémantiques de l'espace des *core assets* sont liées à d'autres attributs de définition de la relation. Le tableau 8.3 présente les contraintes associées à la sélection de chaque propriété sémantique de l'espace des *core assets*. Dans la pratique, des contraintes OCL sont également établies afin de vérifier le comportement souhaité.

Tableau 8.3: Propriétés sémantiques prédéfinies - espace de modélisation.

Nom	*Propriétés associées*
uml-includes	horizontal modeling level, intraconcern, impact none, high criticity level, design
uml-extends	horizontal modeling level, intraconcern, implication, impact none, high criticity level, design
uml-generalization	intraconcern, co-implication, high criticity level, impact none, design
inherits-from	intraconcern, implication, high criticity level, impact none, design
is-instance-of	vertical modeling level, intraconcern, implication, impact none, design
object-association	horizontal modeling level, intraconcern, impact none, run
spem-finishToStart	horizontal, intraconcern, implication, design, activation, high level
spem-finishToFinish	horizontal, intraconcern, implication, design, activation, high level
spem-startToStart	horizontal, intraconcern, implication, design, activation, high level
spem-startToFinish	horizontal, intraconcern, implication, design, activation, high level
has-evolved-to (*is-evoling-to*, *will-evolve-to*, *may-evolve-to*)	horizontal, intraconcern, over time, induction, high level
overrides	horizontal, intraconcern, design, modification, high level

D'autres propriétés se rattachant à des modélisations de systèmes existent, mais n'ont pas fait l'objet d'une attention particulière au cours du développement de l'approche. La propriété à la valeur *NA* (*Non Applicable*) permet de palier la non exhaustivité du langage et d'étendre ponctuellement ce dernier.

8.5 Définition de PLiMoS intention

Le sous-ensemble PLiMoS relatif à l'expression des relations intentionnelles présenté au chapitre précédent (chap. 7) est exprimable par un modèle de spécification de PLiMoS. Cette section présente ce modèle de spécification, permettant une illustration de l'utilisation du langage. La figure 8.3 décrit le contexte de définition afin de le resituer par rapport à l'organisation générale présentée dans la figure 6.10, page 93.

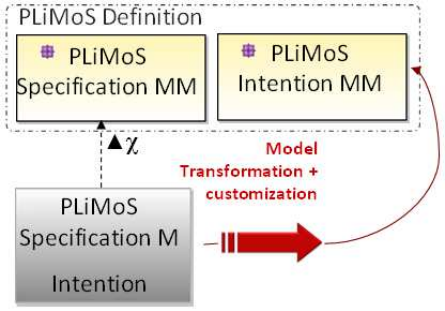


Figure 8.3: Définition de PLiMoS$_{Intention}$ depuis PLiMoS$_{Specification}$

Tout d'abord, la structure de la relation, somme toute relativement simple, prend la définition suivante :

- A = {MuR,ExtModArt1,ExtModArt2} ; (dérivation)
- BR = {MuR} ; (structure relationnelle dyadique)
- Nat = homogeneous ; (dérivation)
- RE = {l,r} ; (deux extrémités distinctes)
- CardSrcMin(l)=0 ; CardSrcMax(l)=-1 ;
- CardTgtMin(l)=0 ; CardTgtMax(l)=1 ;
- CardSrcMin(r)=0 ; CardSrcMax(r)=-1 ;
- CardTgtMin(r)=0 ; CardTgtMax(r)=1 ;
- Lhs(MuR)=l ;
- Rhs(MuR)=r ;
- EAR = {<MuR,ExtModArt1>,<MuR,ExtModArt2>} ;
- MA = {ExtModArt1,ExtModArt2} ;

La propriété sémantique associée n'est pas prédéfinie dans la liste fournie par PLiMoS, elle est donc définie comme suit :

- SI = ROf ;
- Al(ROf) = both ;
- Cr(ROf) = inter ;
- Crit(ROf) = high ;
- Desc(ROf) = "RepresentationOf" ;
- IsDef(ROf) = True ;
- Nat(ROf) = concrete ;
- Time(ROf) = design ;
- Impact(ROf) = none ;
- Merged(ROf) = True ;
- KH(ROf) = False ;

- ER = {∅} ; E = {∅} ; O = {∅} ; OR = {∅} ; DE = {∅} ; InfFrom = {∅} ; HSS = {∅} ; LHSR = {∅} ; RHSR = {∅} ;
- REI = {rei1,rei2} ;
- REIR = {<ROf,rei1>,<ROf,rei2>} ;
- QA = {qaK,qaN,qaI} ;
- QAR = {<ROF,qaK>,<ROf,qaN>,<ROf,qaI>} ;
- PVal = {different,share,sub,super,same,analytical,synthetical,EString} ;
- PValR = {<qaK,different>, <qaK,share>, <qaK,sub>, <qaK,super>, <qaK,same>, <qaN,analytical>, <qaN,synthetical>, <qaI,EString>} ;

Avec les précisions suivantes concernant les interprétations des extrémités :

- SrcN(rei1) = “represent” ; TgtN(rei1) = “src” ; Reif(rei1) = False ;
- SrcN(rei2) = “isRepresentedBy” ; TgtN(rei2) = “tgt” ; Reif(rei2) = False ;

Concernant les attributs de qualités, en voici les descriptions :

- Idt(qaK) = “kind” ; Desct(qaK) = “RepresentationKind” ;
- Many(qaK) = False ; Reified(qaK) = False ;
- Idt(qaN) = “nature” ; Desct(qaN) = “Nature” ;
- Many(qaN) = False ; Reified(qaN) = False ;
- Idt(qaI) = “Intention” ; Desct(qaI) = “Intention Body” ;
- Many(qaI) = False ; Reified(qaI) = True ;

Enfin, quelques différences persistent entre cette définition et le langage présenté dans le chapitre précédent. En effet, le langage d'action ensembliste réifié dans PLiMoS$_{intention}$ explique sa présence par l'historique de sa construction. Afin de jouer le même rôle, une définition depuis PLiMoS$_{specification}$ d'expressions de dérivation basées sur des langages d'action existants (plus puissant) est possible.

8.6 En résumé

Le langage de modélisation PLiMoS$_{specification}$ (Ps) est le sous-ensemble du langage PLiMoS relatif à la définition de *relationships* entre éléments de modèles. Tel qu'introduit antérieurement, le langage promeut une dichotomie entre le canevas de la relation, c.-à-d. son maillage structurel, et ses interprétations sémantiques associées. Ainsi, tout canevas relationnel structurel est associé à une ou plusieurs interprétations sémantiques, elles mêmes adjointes à des relations de modélisation intentionnelles.

Les *relationships* ainsi définies sont caractérisées par une portée, une complexité (elles peuvent posséder des attributs, des gardes et une opérationnalisation), une symétrie, une transitivité, et une navigabilité. Ces entités possèdent un important degré d'expressivité de coordination, sont abstraites, concrètes, voir dérivées et leur impact va de la simple induction à la modification, en passant par l'activation.

Un ensemble de contraintes vérifient des propriétés sémantiques préétablies, qui viennent confiner l'espace des possibles, et spécialisent de la sorte le langage aux *lignes de produits dirigées par les modèles*.

La généricité de PLiMoS$_{specification}$ permet la définition de nombreux langages relationnels, dont PLiMoS$_{intention}$; l'union de ces derniers formant l'ensemble PLiMoS$_{definition}$.

CHAPITRE 9

Opérationnalisation du langage relationnel

"[Le Nouvel Esprit scientifique est] l'esprit humain cherchant à saisir les phénomènes [...] pour en rendre raison, et pour les préparer et les faire naître."

—Gaston Bachelard, *Le Nouvel Esprit scientifique, 1934*

L'ingénierie des *lignes de produits* repose sur la gestion de modèles hétérogènes définissant des artefacts réutilisables et des modèles permettant leur manipulation. La complexité et l'hétérogénéité forte de ces modèles interdépendants entraîne une difficulté notable de gestion de l'information relationnelle et de la cohérence de l'espace de modélisation. Sur la base de la modélisation relationnelle décrite par le langage PLiMoS dans le chapitre précédent, vient s'ajouter une sémantique opérationnelle, permettant d'implémenter des "frameworks" outillés, par exemple pour les transformations et les générations, la visualisation des informations, la vérification de la cohérence.

Ce chapitre traite des activités et opérateurs de PLiMoS en s'intéressant plus particulièrement dans un premier temps, aux transformations autour de $\text{PLiMoS}_{specification}$, puis au langage généré $\text{PLiMoS}_{specialisation}$ et à quatre phases du processus : la modélisation et sa visualisation, la fusion de modèles de variabilité, et la cohérence des modèles. Le chapitre 13 et l'annexe C présentent des détails concernant l'implémentation des opérateurs de ce chapitre. Les éléments exposés ne présentent pas de contributions dans les domaines concernés (transformation, etc.), mais illustre seulement l'opérationnalisation du langage.

Certains éléments de ce chapitre figurent dans les travaux présentés à APSEC 2012 [CCJM12b], ainsi qu'à SIMF 2013 [CC13].

9.1 Autour de PLiMoS specification

$PLiMoS_{specification}$ est un sous-ensemble du métamodèle PLiMoS générique d'expression de modélisation relationnelle. Un modèle de spécification permet, par une génération et une promotion

des concepts, l'obtention d'un métamodèle spécifique $PLiMoS_{specialisation}$. Cette section présente cette génération, ainsi que deux autres opérations essentielles à une pérennité du langage : une migration automatisée des modèles de variabilité, une co-évolution du langage.

9.1.1 Génération de PLiMoS specialisation

La figure 9.1 illustre la description du processus de Génération de PLiMoS$_{specialisation}$. L'éditeur de modèles PLiMoS produit un modèle, intrant du générateur de métamodèle. Le générateur délivre également un modèle $\Delta Mapping$ (cf. section 9.1.1).

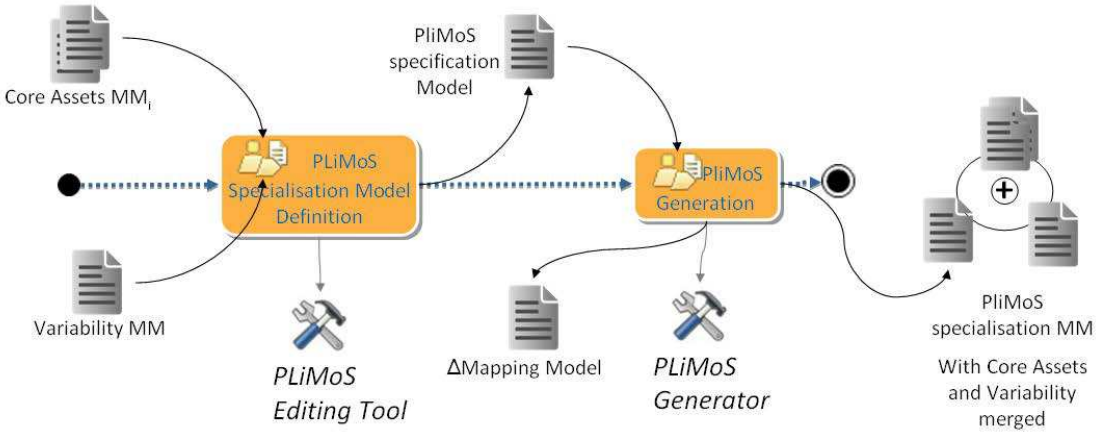


Figure 9.1: Processus de génération des modèles PLiMoS$_{specialisation}$

Lors de la transformation les propriétés présentes sous forme de sélections d'énumération sont traduites en des héritages de classes abstraites (si la propriété *KeepHierarchy* est à vrai). En outre, d'autres paramètres de génération sont disponibles, à l'exemple de la propriété *Merged* de l'entité *SemanticsInterpretation*. Afin d'illustrer ces propos, considérons le cas d'exemple donné section 8.5, la définition de PLiMoS$_{intention}$. La figure 9.2 - A - présente un extrait du métamodèle d'intention obtenu suite à une génération avec le paramètre *Merged=True* (cela ne présente aucun intérêt bien évidemment pour PLiMoS$_{intention}$).

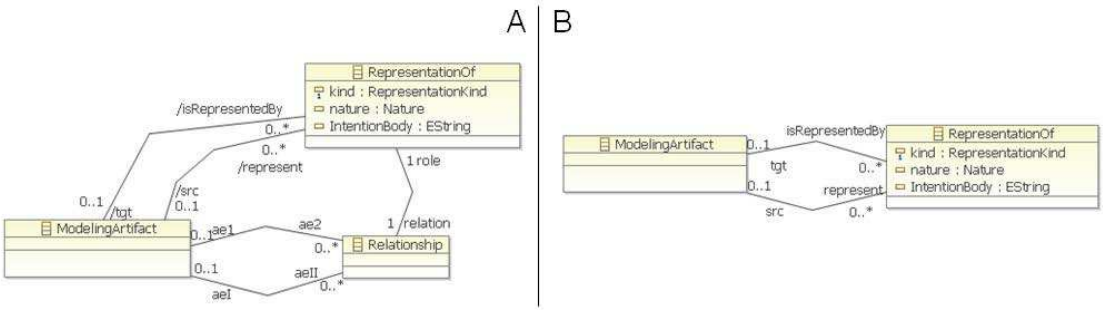


Figure 9.2: Processus de génération des modèles PLiMoS$_{specialisation}$

Autre exemple, la propriété des attributs de qualité *Reified* propose une option de génération, présentée figure 9.2 - B - (à comparer à la figure 7.3), la spécification est donnée ci-après ;

- Idt(qaI) = “Intention” ; Desct(qaI) = “Intention Body” ;
- Many(qaI) = False ; Reified(qaI) = False ;

Enfin, et pour plus de détails, les spécifications des règles de génération sont fournies annexe C.

Vers une migration automatisée des modèles de variabilité

La réalité se présente, en résumé, dans les termes suivants : le langage PLiMoS, fusionné avec des langages de variabilité existants permet d'adjoindre de la précision sémantique à ces derniers. Le besoin subséquent de traduction des modèles d'origine vers le nouveau formalisme et vice versa se fait sentir. La figure 9.3 résume la situation, et représente sous la forme de différence Δ (i) l'ajout de sémantique et donc de concepts au niveau métamodèle, et (ii) la différence de conformité conséquente au niveau modèle.

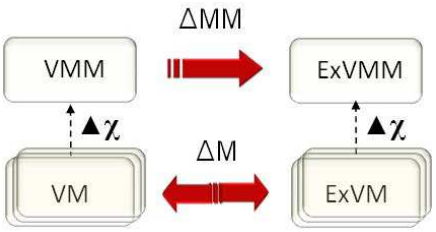


FIGURE 9.3: Variabilité des métamodèles de variabilité

Il est d'un intérêt plus que substantiel de pouvoir récupérer des modèles existants, conformes aux métamodèles de variabilité communément employés (VM). De même il est important de pouvoir exporter des modèles issus de PLiMoS (modèles de variabilité augmentés en capacité $ExVM$ - *Extended VM*), avec immanquablement une perte d'information sémantique liée au manque d'expressivité des langages initiaux.

La transformation est par nécessité bidirectionnelle, mais unilatéralement conservatrice, en effet, en considérant $T_{\Delta_{MM}}$ comme étant la fonction de transformation réalisant la spécification de transformation Δ_{MM} :

$$T^{-1}_{\Delta_{MM}}(T_{\Delta_{MM}}(VM)) = VM; T^{-1}_{\Delta_{MM}}(T_{\Delta_{MM}}(ExVM)) \neq ExVM;$$

La figure 9.4 présente le processus de migration. Le model Δ*Mapping* établit lors de la génération de PLiMoS$_{specialisation}$ délivre les correspondances à l'interpréteur de *mapping* (*PLiMoS Migration Tool*) afin de générer le modèle de variabilité étendu (*ExVM*, ou *ExVariability M* sur la figure). Le flux peut être inversé pour T^{-1}.

La réalisation du Δ*Mapping* est conséquente à la définition de relation par défault (*isDefault=True*) dans PLiMoS$_{specification}$, homonymes de relations existantes dans les métamodèles d'origine.

Note : l'opérationnalisation de la migration n'est en rien spécifique aux modèles de variabilité, cependant, dans le contexte des *lignes de produits*, ce besoin de migration est omniprésent, tant et si bien que l'outillage en a pris la dénomination.

Une co-évolution des modèles PLiMoS$_{specialisation}$

La figure 9.5 illustre le processus de co-évolution des modèles PLiMoS. Les modifications à apporter pour permettre de transformer un modèle en version 1 (Model V1) en version 2 sont

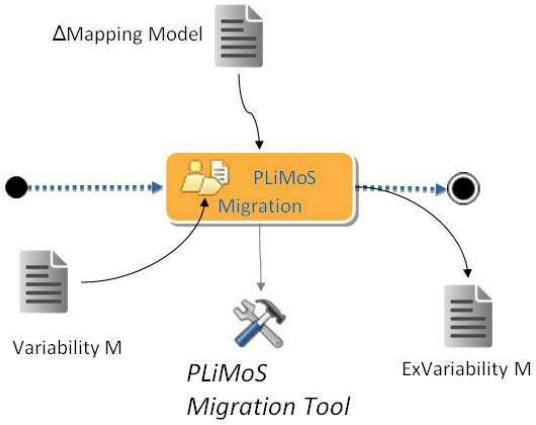


Figure 9.4: Processus de migration des modèles de variabilité

fournies par l'analyse de la similitude entre MMV1 et MMV2 (plus la distance structurelle et sémantique une petite, plus le résultat sera probant).

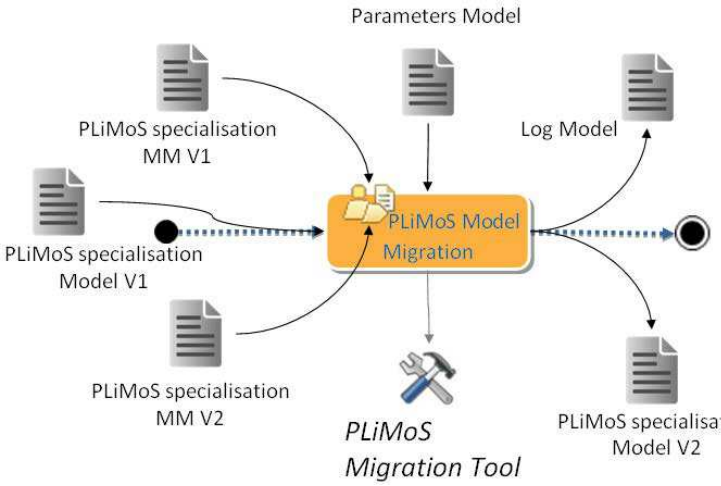


Figure 9.5: Processus de co-évolution des modèles PLiMoS$_{specialisation}$

L'alignement entre deux entités est souvent rompu quand l'une des entités évolue. C'est alors dans le but de gérer l'évolution conjointe de ces deux entités qu'une gestion de l'alignement est entreprise. Cependant, l'évolution d'une entité entraîne souvent une rupture de la relation d'alignement. Dès lors, son impact sur les autres entités doit être analysé. Quatre caractéristiques de connexion existent : indépendance, interdépendance, dépendance, et double dépendance. Chaque type est défini selon la direction du lien de dépendance entre les entités qui évoluent. La rupture de ces derniers correspond à un traitement de gravité graduel. Par ailleurs, il est à noter que le nombre d'entités alignées se réduit à deux (dyadique).

Le modèle de paramètre de la figure 9.5 peut, au besoin, fournir des équivalences sémantiques entre entités afin de faciliter la conversion des modèles. Il détermine également l'échelonnage de

la gravité des ruptures, et peut accorder la validation de pertes d'informations. Enfin, le modèle de *Log* résultant renseigne exhaustivement sur les actions réalisées lors de la transformation.

9.2 Autour de PLiMoS specialisation

Les *relationships* définies et spécialisées sont caractérisées par des propriétés sémantiques précises et trouvent leur utilité à travers plusieurs usages. La section présente des opérateurs de visualisation, de fusion et de vérification de la cohérence.

9.2.1 Compréhension et visualisation

La section présente des opérateurs liés à la compréhension et à l'amélioration de cette dernière pour apporter des moyens de visualisation plus élaborés.

L'ensemble des éléments requis d'un élément e (*mandatory(e)*)

L'ensemble des éléments requis (noté $R(e)$) par un élément particulier e, soit *mandatory(e)*, est obtenu par la création d'une fermeture transitive *(transitive closure)* depuis l'élément ciblé, en utilisant les relations d'implication et de co-implication, et en laissant de côté les éléments optionnels et les variants non impliqués.

La commonalité d'un élément e (*common(e)*)

La commonalité d'un élément donné e est formée par ses descendants obligatoires (relations de décomposition hiérarchiques de type généralisation, classification, agrégation et composition, associées par un opérateur *And*, les éléments des *Xor* et *Or* en étant exclus). La commonalité d'un élément est l'intersection de l'ensemble de ses descendants hiérarchiques et de l'ensemble de ses éléments requis. Considérons $H(e)$ comme étant l'ensemble des descendants hiérarchiques d'un élément, l'ensemble des éléments communs $C(e)$ s'exprime comme suit :

$$C(e) \triangleq H(e) \cap R(e)$$

Un opérateur de commonalité large $C_w(e)$ est défini, prenant en compte également les relations liées à un opérateur *Or*.

La commonalité de la *ligne de produits*

La commonalité de la *ligne de produits* C_{LdP} est définie dans le cas de modèles de variabilité de type arbre comme la commonalité de l'élément racine (r), soit $C(r) : C_{LdP} \triangleq C(r)$.

Quelques autres opérateurs

Pour permettre une réalisation de graphes et de chemins de mise en relation d'artefacts :

- *connected(e)* : retourne l'ensemble des éléments connectés directement (séparé par une relation) à un élément donné ;
- *connected(e,nb)* : retourne l'ensemble des éléments connectés à un élément donné par un degré relationnel de *nb* ;
- *relationships(e)* : retourne l'ensemble des relations connectées en direct à un élément donné ;
- *relationships(e,nb)* : retourne l'ensemble des relations connectées à un élément donné par un degré relationnel de *nb*, l'ensemble est ordonné en sous-ensembles de degré relationnel (ensemble des relations de rang 1, puis de rang 2, etc.) ;

- *areConnected(e1,e2)* : vérifie s'il existe un chemin relationnel entre deux éléments $e1$ et $e2$ (renvoie vrai le cas échéant) ;
- *relationships(e1,e2)* : renvoie un ensemble de relations définissant un chemin reliant $e1$ à $e2$ si celui si est défini, ne renvoie rien sinon ;
- *connected(e1,e2)* : renvoie un ensemble d'éléments figurant sur le chemin reliant $e1$ à $e2$ si celui-ci est défini, ne renvoie rien si le lien est direct, et rien si les éléments ne sont pas liés ;

Des opérateurs d'entités sur les relations :

- *setValidity(hs)* : change l'état de validation d'un élément ;
- *setValue(att,val)* : mise à jour de la valeur d'un attribut (cas de modèles de variabilité possédant des attributs, ou de *core assets* liés par des relations de modifications) ;
- *setSelection(e,enumlit)* : mise à jour de la sélection d'un élément variable pour la configuration de produits par les littéraux (selected, undecided, eliminated, conflict).

Enfin, et pour améliorer la compréhension et l'analyse, des opérateurs de type métrique renvoient :

- Le nombre d'éléments référencés par type de relation ;
- Le nombre d'éléments de tel type connecté ;
- Le nombre d'interdépendances entre deux éléments ;
- Le nombre de relations composant le plus court chemin entre deux éléments ;

9.2.2 Fusion des modèles de variabilité

La composition de modèles de variabilité représentant des préoccupations diverses s'apparente généralement à de la fusion de modèles et permet une intégration des informations. L'objectif est d'obtenir une perspective unifiée, de permettre une meilleure compréhension des interactions entre les éléments des différents modèles, et de dérouler ainsi différents types d'analyses (par ex. de la vérification de cohérence).

La fusion se retrouve également sous le terme réconciliation de modèles de variabilité. Les techniques de gestion des modèles mettent en avant un certain nombre d'opérations pour manipuler les modèles, tels que par exemple le *match*, le *merge* et le *check property*. D'une part, le *match* fournit un moyen de découvrir les relations entre des modèles : comparaison de variants, identification d'incohérence, etc. D'autre part, le *merge* fournit un moyen d'obtention d'une perspective unifiée, permet de mieux comprendre des interactions entre les modèles, et éventuellement de réaliser des analyses telles que la validation et la vérification.

Les critères de fusion (ϕ) sont les suivants :

- la complétude : tout concept (c) apparaissant dans un modèle d'entrée ($Intrant$) doit être présent dans le modèle résultant ($Extrant$), soit
  $\forall c \in Intrant, \exists c' \in Extrant | \phi(c) = c'$;
- La non redondance : si un concept apparaît dans plusieurs modèles d'entrée, une copie unique doit apparaître dans le modèle résultant, $\forall (c_i \in Intrant_i, c_j \in Intrant_j, i \in \mathbb{N}, j \in \mathbb{N} | c_i \equiv c_j \wedge i \neq j) \nexists c'_i \neq c'_j | \phi(c) = c'$;
- la minimalité : l'opération n'introduit pas d'opération nouvelle : $Extrant = \cup \phi(c), c \in Intrant$;
- la totalité, l'opération est réalisable pour n'importe quel nombre de modèles d'entrée ;
- la justesse, l'opération supporte l'expression et la préservation des propriétés sémantiques.

En considérant les différents modèles de variabilité à fusionner comme des ensembles d'éléments variables EV, le modèle résultant (vu tel un ensemble EVR) est défini comme l'intersection des EV ($[\![\cdot]\!]$ représente l'ensemble des configurations valides), soit :

$$[\![EVR]\!] \triangleq \bigcap_{i=1}^{n} [\![EV_i]\!]$$

L'approche propose une définition d'un axiome de compositionnalité pour des modèles de variabilité, sur la base des relations d'équivalence et de similarité (cf. section 8.4.1). Elle offre de ce fait, une composition partielle ou globale des modèles de variabilité, avec filtrage des préoccupations possibles.

L'opérateur de fusion représente un cas simple, car il concerne des modèles homogènes (de *features*, ou autres), avec une spécification explicite du chevauchement. L'interprétation de l'équivalence est la suivante : L'opérateur $\oplus$ est défini tel que :

$$\oplus \triangleq \forall x \in A, y \in B \cdot x = y \Rightarrow C = (C \cup \{x\}) \wedge (C \cup \{y\})$$

La stratégie de composition de similarité est la suivante : un renommage d'un des éléments pour permettre l'alignement, le renommage par défaut soutenant l'abstraction (propagation du plus au moins abstrait). Dans la pratique, la fusion des modèles de variabilité entraîne une création d'un nouveau modèle.

Note : la fusion de *core assets* lors de la réalisation d'un produit peut suivre le même principe. Le *match* et l'identification des relations d'équivalence sémantique se doivent d'être automatisés (par exemple sur la base de règles de composition du langage de modélisation ciblé, cf. section 11.5.1).

9.2.3 Cohérence des modèles

Maintenir un alignement opérationnel revient à conserver une cohérence sémantique et structurelle entre des modèles hétérogènes, d'abstraction et de granularités différentes.

Les modèles de l'espace de modélisation de la *ligne de produits* associés par le langage PLiMoS sont de deux grandes natures, entrainant une gestion de la cohérence sur deux plans distincts :

1. La cohérence dans le cadre de l'IDM : avec en premier lieu une conformité aux langages (et métamodèles), ainsi qu'aux règles OCL additionnelles (notamment dans les modèles de variabilité). Au delà des considérations sur l'ensemble coercitif de cohérence dans un IDM, des contraintes de conception sont susceptibles d'être définies pour respecter un style architectural et imposer des restrictions sur les possibilités d'évolution de la LdP (pour plus de détails, voir chapitre 10). Par exemple, pour un style d'architecture en couches, la contrainte peut décrire qu'une couche de haut niveau peut utiliser une de ses semblables de plus bas niveau mais pas l'inverse.

2. La cohérence dans le cadre de la LdP : avec une vérification des règles de bonne formation de la variabilité, une cohérence logique des produits (de type sat) et des *core assets* (gérés par l'IDM dans la perspective de *safe composition*).

La suite du paragraphe s'attache à décrire la seconde cohérence, celle de la *ligne de produits*.

L'objectif est d'inscrire une procédure opératoire de validation, afin de rendre possible les tests qui sont mis en œuvre au plan formel pour interroger les formules correspondantes. Encore faut-il que le caractère normatif des règles de translation ait été circonscrit avec netteté, et qu'aucun postulat de signification ne soit importé dans la procédure.

Il est donc primordial d'établir des équivalences propositionnelles depuis le "monde" IDM et de fournir une sémantique translationnelle au langage PLiMoS. Conséquemment, des règles d'inférence sont définies pour les langages de variabilité (FM, OVM, voir ci-après), et dès lors,

concernant le langage PLiMoS, il s'agit d'effectuer une translation en négligeant les nuances de sens des diverses interprétations sémantiques.

L'objectif de la validation est de circonscrire le champ d'application d'une formule donnée sur une distribution de valeurs de vérité. La formule est rendue vraie (ou satisfiable) s'il est démontrable qu'elle est vraie pour toutes les assignations qui lui sont données. L'approche se place en utilisateur de techniques éprouvées de vérification de satisfiabilité des modèles, dans le cas présent, à l'aide de solveurs SAT[1] [Man02, MWC09, TBK09]. Des exemples d'opérations classiques associées sont : la vérification de satisfiabilité, la vérification qu'une configuration donnée est valide au vu des contraintes, le calcul de la commonalité, la détection de la présence d'éléments fantômes (*dead nodes*) [BCTS06].

Les opérations sont définies comme suit :

- *Satisfiabilité* : un modèle de variabilité vm conforme à un langage $\mathcal{L}$ est satisfiable si une configuration valide peut être établie - $[\![vm]\!]_{\mathcal{L}} \neq \emptyset$; cette opération renseigne sur la cohérence du modèle de variabilité et des *relationships* associées ;
- *Validité d'une configuration* : une configuration c est dite valide si elle satisfait les contraintes du modèle de variabilité vm - $c \in [\![vm]\!]_{\mathcal{L}}$;
- *Découverte de la commonalité* : les éléments communs $CE \subset E$, avec E désignant l'ensemble des éléments, sont tels que $\forall c \in [\![vm]\!]_{\mathcal{L}} \cdot CE \subseteq c$

Règles de translation vers l'espace propositionnel

La sémantique translationnelle est associée à PLiMoS et donc aux modèles de variabilité associés. Les cas d'études traités se limitent à l'utilisation de modèles de *features* et de modèles de type OVM (introduit section 3.2.4), en conséquence, sont données ci-après les règles de translation de ces deux catégories de modèles.

Tout d'abord, concernant les modèles de *features* : Soit c un *feature* composite, S représente l'ensemble des sous-*features* s de ce dernier, avec s_p désignant un sous-*feature* particulier ($p \in N = \{1, \ldots, n\}$, et $n = card(s)$. De la même manière, définissons $q \in N$). De plus, désignons par f un *feature* quelconque. Le tableau 9.1 présente un résumé des équivalences entre les opérateurs des modèles de *features* et la logique propositionnelle. Pour rappel, l'opérateur $V_{i,j}$ est l'opérateur générique logique, les diverses valeurs de i et j sont observées dans le tableau.

Ensuite, de manière similaire, des équivalences logiques sont établies entre le langage OVM et les formules propositionnelles (cf. tableau 9.2). Dans ce tableau, vp représente un point de variation, v in variant ($v \in V$, ensemble des variants de vp, avec $card(V) = n$).

Enfin, concernant PLiMoS et ses relations, un nombre non unitaire de propriétés sémantiques peut être associé à une même structure relationnelle ; ces dernières, si elles possèdent un des attributs de coordination (*implication, co-implication, exclusion*) identiques sont équivalentes d'un point de vue propositionnel. Concernant l'attribut de coordination propositionnelle, il est primordial que toute relation ayant pour vocation à être interprétée en logique soit renseignée. Par ailleurs, les niveaux de criticité des relations (*high, medium, low*) rentrent en compte comme paramètre du module de translation, permettant une certaine sélectivité.

Dans la pratique, les propositions logiques doivent être converties dans une forme conjonctive normale (*conjunctive normal form (CNF)*) pour être absorbées par le *solveur* ; des algorithmes reconnus le permettent. Ne constituant pas le propos de la thèse, l'implémentation de la translation est réalisée de manière simple par l'implantation d'un visiteur permettant un parcours récursif dans les graphes ; aucune optimisation n'a fait l'objet d'études approfondies.

[1] D'autres solveurs peuvent être utilisés à ces fins, par ex. CSP [WSB+08], ou BDD packages [MWCC08]

Tableau 9.1: Modèles de *features* et formules propositionnelles

Modèles de *features*	Formules propositionnelles
Optional, $V_{0,1}$	$s_p \Rightarrow c$
$V_{0,n}$	$\bigvee_{p \in N} s_p \Rightarrow c$
Xor, $V_{1,1}$	$\left(c \Leftrightarrow \bigvee_{p \in N} s_p\right) \wedge \left(\bigwedge_{p,q \in N \mid p<q} (\neg s_p \vee \neg s_q)\right)$
Or, $V_{1,n}$	$c \Leftrightarrow \bigvee_{p \in N} s_p$
And, $V_{n,n}$	$\left(c \Rightarrow \bigwedge_{p \in N} s_p\right) \wedge \left(\bigvee_{p \in N} s_p \Rightarrow c\right)$
$V_{i,j}$, $1 \leqslant i \leqslant j < n$	$\bigvee_{M \subset S \mid i \leqslant Card(M) \leqslant j} \left(\left(c \Leftrightarrow \bigwedge_{m \in M} s_m\right) \wedge \left(\bigwedge_{r \in M, s \in N/M} (\neg s_r \vee \neg s_s)\right)\right)$
$V_{0,j}$, $1 < j < n$	$\bigvee_{M \subset S \mid i \leqslant Card(M) \leqslant j} \left(\left(c \Rightarrow \bigwedge_{m \in M} s_m\right) \wedge \left(\bigwedge_{r \in M, s \in N/M} (\neg s_r \vee \neg s_s)\right)\right)$
Implication	$f_1 \Rightarrow f_2$
Co-Implication	$f_1 \Leftrightarrow f_2$
Exclusion	$\neg (f_1 \wedge f_2)$

Tableau 9.2: Modèle OVM et formules propositionnelles

Entités d'OVM	Formules propositionnelles
mandatory	$vp \Leftrightarrow v$
optional	$v \Rightarrow vp$
alternative choice $(min = 1, max = 1)$	$\left(vp \Leftrightarrow \bigvee_{p \in N} v_p\right) \wedge \left(\bigwedge_{p,q \in N \mid p<q} (\neg v_p \vee \neg v_q)\right)$
alternative choice $(min = 1, max = n)$	$vp \Leftrightarrow \bigvee_{p \in N} v_p$
alternative choice $(min = n, max = n)$	$\left(vp \Rightarrow \bigwedge_{p \in N} v_p\right) \wedge \left(\bigvee_{p \in N} v_p \Rightarrow vp\right)$
alternative choice $(min = i, max = j)$, $1 \leqslant i \leqslant j < n$	$\bigvee_{M \subset V \mid i \leqslant Card(M) \leqslant j} \left(\left(vp \Leftrightarrow \bigwedge_{m \in M} v_m\right) \wedge \left(\bigwedge_{r \in M, s \in N/M} (\neg v_r \vee \neg v_s)\right)\right)$
alternative choice $(min = 0, max = j)$, $1 < j < n$	$\bigvee_{M \subset V \mid i \leqslant Card(M) \leqslant j} \left(\left(vp \Rightarrow \bigwedge_{m \in M} v_m\right) \wedge \left(\bigwedge_{r \in M, s \in N/M} (\neg v_r \vee \neg v_s)\right)\right)$
requires_V_V	$v \Rightarrow v$
requires_V_VP	$v \Rightarrow vp$
requires_VP_VP	$vp \Rightarrow vp$
excludes_V_V	$\neg(v \wedge v)$
excludes_V_VP	$\neg(v \wedge vp)$
excludes_VP_VP	$\neg(vp \wedge vp)$

Exemple d'application à une décomposition de modèles de *feature*

Considérons le cas particulier introduit dans la section 6.1.1, avec une séparation de la variabilité du contexte CVM, du problème EVM, et de la solution IVM (pour rappel il s'agit de la figure 6.1, page 86). Dans l'exemple suivant, chaque type est représenté par un modèle de *features*.

La cohérence de ces modèles pourrait être envisagée séparément, néanmoins la vérification de cohérence doit concerner également la vérification de l'absence de relations conflictuelles dans et entre les différents modèles. L'incohérence dans les différents modèles de *feature* correspond à un état dans lequel deux ou davantage d'éléments appartenant à différents modèles font des affirmations sur certains aspects du système qu'ils décrivent qui ne sont pas conjointement satisfiables.

Considérons l'ensemble des configurations de variabilité possibles (ev_{CVM}) de l'environnement $[\![CVM]\!]_{\mathcal{L}}$. Soit, également, $[\![EVM]\!]_{\mathcal{L}}$ l'ensemble des produits (ev_{EVM}) que le management des *lignes de produits* décide d'offrir. Enfin $[\![IVM]\!]_{\mathcal{L}}$ symbolise l'ensemble des produits (ev_{IVM}) que la plate-forme autorise à construire. Les produits (p) vérifient une configuration valide ($ev \in [\![VM]\!]_{\mathcal{L}}$), les éléments variables ev se traduisant en pratique par des *features*, variants et points de variation. Dans la suite, la notation $|_x$ représente une projection sémantique sur un ensemble d'éléments variables (E) : $Z|_X = \{y \cap X | y \in Z\}$.

La cohérence de ces modèles peut être dans un premier temps envisagé séparément (satisfiabilité séparée), des vérifications combinées sont également réalisables et sont exprimées comme suit :

- *V1 :* Assouvissement de la variabilité du contexte. Existe-t-il des contraintes de contexte (ou variabilité externes environnementales) non réalisables ? Une combinaison de contraintes produits du *CVM*, $ev_{CVM} \in [\![CVM]\!]$, est réalisable s'il existe des liens d'induction environnementale permettant d'assouvir sa variabilité, soit : $ev_{CVM} \in ([\![EVM]\!]|_{Ecvm} \cup [\![IVM]\!]|_{E_cvm})$. Les éléments non réalisables sont donnés par l'ensemble $[\![CVM]\!] \setminus ([\![EVM]\!]|_{Ecvm} \cup [\![IVM]\!]|_{Ecvm})$, qui ne doit être vide.
- *V2 :* Réalisabilité de la *ligne de produits*. Existe-t-il des produits proposés non réalisables par la plate-forme de conception ? $ev_{EVM} \in [\![EVM]\!]$ est réalisable s'il possède un lien de réalisation, soit $ev_{EVM} \in [\![IVM]\!]|_{Eevm}$.
- *V3 :* Identité de réalisation. Existe-t-il une même combinaison de l'IVM réalisant plusieurs combinaisons de l'EVM ? Il est exprimé comme suit, $(ev_{e1} \cup ev_i) \in [\![IVM]\!] \wedge (ev_{e2} \cup ev_i) \in [\![IVM]\!] \wedge (ev_{e1} \neq ev_{e2})$.
- *V4 :* Commonalité de la *ligne de produits*. Existe-t-il de la commonalité entre les produits réalisables par la plate-forme de conception ($\bigcap [\![IVM]\!]|_{Eevm} \setminus \bigcap [\![EVM]\!]$) ?
- *V5 :* Présence d'éléments variables fantômes. Existe-t-il des éléments non réalisés et non satisfiables ? ($E_{EVM} \setminus \bigcup [\![IVM]\!]|_{Eevm}$.
- *V6 :* Multiplicité de réalisation. Existe-t-il plusieurs combinaisons de l'IVM réalisant une même combinaison de l'EVM ? Cet état de fait n'est pas à proscrire, il peut résulter du fait que du non fonctionnel n'apparait pas forcement dans l'EVM, relatif au coût, à la performance, à la sécurité, etc. Il est exprimé comme suit, $(ev_e \cup ev_{i1}) \in [\![IVM]\!] \wedge (ev_e \cup ev_{i1}) \in [\![IVM]\!] \wedge (ev_{i1} \neq ev_{i2})$.

9.3 En résumé

L'opérationnalisation de PLiMoS a trait à ses deux facettes, spécification et spécialisation. Dans le premier cas, il s'agit de mettre en œuvre la mécanique de définition du langage dans un environnement MOF, ce qui se traduit par la réalisation de transformations de modèles. Dans le second cas, l'opérationnalisation touche aux divers usages du langage relationnel final, à savoir : l'édition et la visualisation, la fusion, la vérification de cohérence.

Troisième partie

Évolution incrémentale de la modélisation des *lignes de produits*

"It is not the most intellectual of the species that survives ; it is not the strongest that survives ; but the species that survives is the one that is able to adapt to and to adjust best to the changing environment in which it finds itself . . . so says Charles Darwin in his "Origin of Species"[. . .]"

— Megginson, L. C., *Key to Competition is Management, 1964*

CHAPITRE 10

Considérations architecturales et précision du périmètre de la LdP

"Une grande époque vient de commencer.
Il existe un esprit nouveau.
Il existe une foule d'œuvres d'esprit nouveau ; elles se rencontrent surtout dans la production industrielle.
L'architecture étouffe dans les usages.
Les "styles" sont un mensonge.
Le style, c'est une unité de principe qui anime toutes les œuvres d'une époque et qui résulte d'un esprit caractérisé.
Notre époque fixe chaque jour son style.
Nos yeux, malheureusement, ne savent pas le discerner encore."

—Le Corbusier, *Vers une architecture, 1924*

Un logiciel a comme caractéristique essentielle, qui plus est concernant une *ligne de produits* réactive, d'être continuellement soumis à des changements (évolution des besoins et du contexte). Ainsi, une des propriétés requises est la maîtrise de l'impact d'un changement afin de permettre une certaine anticipation de celui-ci. Cette caractéristique est dans ce cas souvent étroitement liée au caractère modulaire de l'application. Ce chapitre propose l'étude d'une mise en œuvre du principe d'anticipation du changement dans le cadre d'une *ligne de produits*, par l'application d'un patron d'architecture. Les bénéfices d'une telle application sont soulignés par de nombreux travaux, par ex. [GFA98, CN01, vZ02, ABB+02], mais la mise en pratique de ces patrons ainsi que l'impact sur l'espace de modélisation de la *ligne de produits* ne sont généralement pas exprimés.

L'observation de Bengtsson figurant ci-après souligne l'importance d'expliciter les causes et logiques de l'organisation structurelle des développements. La partie II insistait sur les relations intentionnelles, ce chapitre montre l'importance d'avoir un patron d'architecture corrélé à la modélisation de l'espace de modélisation.

> *"[...]a brilliant team of software engineers may still be able to do a good job with a poor software architecture. On the other hand, a perfect software architecture may lead to unacceptable results in the hands of inexperienced software engineers that fail to understand the rationale behind the software architecture."*
>
> Bengtsson [Ben02](traduction p. 207)

Ce chapitre introduit le besoin d'avoir une architecture ouverte de la Ligne de Produits (section 10.1), puis propose une discussion sur la sélection du motif d'architecture à appliquer en fonction de critères de classification (section 10.2). Les deux sections suivantes introduisent le patron d'architecture *ABCDE* (section 10.3), issu de réflexions sur les développements de l'équipe d'accueil de Thales, et en discute les bénéfices (section 10.4). Enfin, une connexion avec le langage PLiMoS introduit dans la partie précédente est réalisée, visant à intégrer les différents artefacts de l'espace de modélisation (10.5).

L'intégration du motif architectural ABCDE en tant que plate-forme conceptuelle dans l'espace de modélisation est décrite en détails dans la partie validation de la thèse (section 12.3), car appliquée au cas particulier de Thales.

Certains éléments de ce chapitre figurent dans les travaux présentés à la conférence sur les architectures logicielles CAL 2012 [CCMJ12].

10.1 Une architecture de *ligne de produits* ouverte

Au moment de créer une architecture logicielle pour un système donné, il est important de connaître les concepts permettant à l'architecture de survivre et d'évoluer. Certaines architectures sont plus à même de supporter l'évolution que d'autres. Afin de créer une architecture durable, il est important d'être capable d'anticiper certains types de changements à même de se produire. Il apparaît donc légitime de s'interroger sur les spécificités nécessaires au patron d'architecture afin qu'il facilite l'évolution de la PLA.

La section 4.2 met en exergue que le patron d'architecture doit être dédié à un type d'applications. Les questions sous-jacentes à la sélection d'un patron sont :

1. Comment savoir si un patron d'architecture donné est le meilleur pour un type de développements à réaliser ?
2. Ce patron offre-t-il de bonnes capacités d'évolution ?

La deuxième question trouvera sa réponse dans la section conclusive de ce chapitre. Revenons dans un premier temps à la première question énoncée, sa réponse est fréquemment pratique et obtenue *a posteriori*, les patrons trouvant naissance à partir de bonnes pratiques propres au développement adressé. En effet, d'un point de vue ascendant (*bottom-up*), les patrons sont des structures récurrentes d'un ensemble de solutions. Dans notre cas, le patron proposé par la suite (nommé ABCDE) résulte d'un consensus de vision sur la problématique des développements de l'équipe de rattachement de Thales ; ce patron va être ensuite appliqué au contexte des LdPs.

Cependant, le recul nécessaire n'est pas toujours possible, et il est intéressant d'observer les critères de sélection d'un patron existant, car, d'autre part, et d'un point de vue descendant (*top-down*), les patrons sont des gabarits qui représentent une vision consensuelle sur un problème spécifique. Une recherche rapide met en évidence pléthore de résultats de listes de patrons logiciels. Par exemple, le site *Handbook of Software Architecture* [4] catalogue plus de deux milles patrons (de diverses granularités bien entendu). Au vu de ce panel de connaissance disponible il serait aisé d'arriver à la conclusion qu'il n'y a plus qu'à appliquer un ou des patrons pour résoudre le problème adressé. La réalité n'est pas si simple et la question de la détection et de la sélection d'un (des) bon(s) patron(s) se pose.

10.2 La sélection d'un patron d'architecture

La littérature propose peu de méthodologie montrant comment mettre en application et tirer pleinement profit de multiples patrons dans une solution, ce qui entraine fréquemment une utilisation hasardeuse et peu stratégique des patrons.

Les patrons sont généralement identifiables par un certain nombre de critères, ceux-ci concernent la structure, l'intention, ou l'applicabilité. Une représentation héritée du *Gangs Of Four* propose une description par le triptyque contexte, problème, solution.

L'intérêt de cette section réside en la proposition de critères permettant d'amener à la sélection d'un motif de conception. Les critères de classification des patrons dans la littérature sont nombreux et l'objectif de la section n'est pas de proposer une classification exhaustive des patrons mais de rappeler certains critères importants.

Le premier critère de classification est la granularité, décrite dans l'ouvrage de Buschmann *et al.* [BMR+96], à savoir : architecture, *design*, et *idioms*. Les travaux décrivent 17 patrons dédiés à la résolution de problèmes distincts. Une classification peut donc également être réalisée suivant l'objectif (ou la catégorie de problème), distinguant ainsi les patrons :

- La décomposition structurelle ("from mud to structure") : *Layers, Blackboard, Pipes and Filters,* et *Whole Part* ;
- Les systèmes distribués : *Broker, Forwarder- Receiver*, et *Client-Dispatcher-Server* ;
- Les systèmes interactifs : *Model-View-Controller* et *Presentation-Abstraction-Control* ;
- Les systèmes adaptatifs : *Microkernel, Reflection* ;
- L'organisation des tâches : *Master Slave* ;
- L'accessibilité au service : *Proxy* ; la gestion des ressources : *Counted Pointer, Command Processor*, et *View Handler* ; la communication : *Publisher-Subscriber* ;

Appliqué à la Ligne de Produit, Gomaa [Gom04] propose la distinction suivante :

- Patrons d'architecture structurels : *Broker, Centralized Control, Client/Agent/Server, Client/-Server, Distributed Control, Hierarchical Control Kernel, Layers of Abstraction* ;
- Patrons d'architecture de communication : *Asynchronous Message Communication, Asynchronous Message Communication with Callback, Bidirectional Asynchronous Message Communication, Broadcast, Broker Forwarding, Broker Handle, Discovery, Negotiation, Subscription/Notification, Synchronous Message Communication with Reply, Synchronous Message Communication without Reply* ;
- Patrons d'architecture de transaction : *Compound Transaction, Long-Living Transaction, Two-Phase Commit Protocol* ;

L'objectif premier est d'organiser la PLA, le choix se porte donc naturellement vers des motifs architecturaux structurels qui sont introduits dans la LdP pour réaliser une plateforme conceptuelle et organiser l'espace de modélisation ainsi que les modèles le constituant.

10.3 Le patron d'architecture ABCDE pour les *lignes de produits*

Dans les applications traitées par Thales, la variabilité de l'évolution provient pour une part importante des acteurs externes au système. Les changements associés à une description d'architecture logicielle qui concernent l'adaptation de l'architecture aux nouveaux besoins d'évolution externe sont considérés comme le principal problème.

Le principe appliqué part du constat que dans la plupart des applications, les modules qui portent la logique fonctionnelle de l'application, appelée aussi logique "métier", sont construits directement au dessus des modules de bas niveau appelés aussi modules "techniques". L'anticipation du changement consiste dans le domaine de l'architecture logicielle à construire un modèle

conscient du besoin permanent de changement du système, capable de spécifier et de "cadrer" ces changements, c'est-à-dire de préciser la variabilité de l'architecture.

Le patron d'architecture *ABCDE*, issu des réflexions et développements à Thales est ici adapté à la *ligne de produits*.

Cette section s'attache à décrire le patron *ABCDE* suivant le schéma classique contexte, problème, solution (avec une description statique et dynamique de la solution).

10.3.1 Dénomination

Le patron *ABCDE* prend sa dénomination des couches le constituant (présenté plus loin en section 10.3.4). Ce patron intègre la classification existante de patrons : il s'agit bien d'un patron d'architecture, de type "from mud to structure"[BMR+96], à l'instar des patrons "Layer", "Blackboard" et autres confrères.

10.3.2 Contexte

La situation organisationnelle dans laquelle le patron peut s'appliquer est la suivante : toute organisation intéressée par la réutilisation en Ligne de produits de systèmes réactifs.

Plus spécifiquement, il est à noter que l'importance déjà forte d'une architecture pour un système logiciel temps réel est accrue pour une Ligne de Produit Logiciel, donnant encore plus de sens à une explicitation des contraintes d'architecture par l'application d'un patron.

En outre, les bénéfices de réutilisation apportés par une *ligne de produits* peuvent être rentables pour des approches autres que la production de masse, notamment en considérant que les LdPs peuvent être sujettes à une évolution réactive. Cette approche est réalisable uniquement si l'architecture est bien définie et ouverte.

Dans le cas particulier de Thales et des applications de défense, la combinaison de ces deux derniers paramètres, applications temps réel et *ligne de produits* réactive, rend primordiale l'application d'un patron d'architecture à la LdP. Les systèmes de défense concernés suivent le fonctionnement global introduit ci-après. Les systèmes tirent parti de capteurs et d'actuateurs (radars, consoles, ...) et leur principale plus-value réside dans le traitement par des algorithmes spécifiques des données d'entrée.

10.3.3 Problème

Quelle est la part des efforts d'adoption de la *ligne de produits* qui doit être accomplie ? Ces efforts sont bien évidemment relatifs aux actifs existants et à l'architecture de départ. La modélisation de l'espace de modélisation de la *ligne de produits* est un pré-requis, mais ne propose, ni ne présuppose, d'aucune organisation architecturale ; ce point est à traiter avec le patron. Une sélection de patron adéquat est primordiale.

10.3.4 Solution

La logique de la sélection et de l'application d'un patron d'architecture est cruciale pour le personnaliser à un problème donné. Le patron dédié *ABCDE* fait le choix d'une séparation des préoccupations suivant une découpe horizontale (en couches) et verticale (en partitions).

Description statique

Inspiré des patrons de la littérature BCED [Mac01] et PAC [Cou87], *ABCDE* est adapté pour se rapprocher des applications cibles orientées traitement. Le patron en couche *ABCDE* propose

une inversion des couches "Data" et "Entity" pour renforcer le cœur de métier (par rapport à des applications orientées données) et l'explicitation de la couche Acteur. Les systèmes ciblés sont de type contrôle-commande, et des actions et traitements en sont la finalité, à l'inverse des systèmes transactionnels de gestion dont la finalité est de traiter et consulter de l'information (avec une persistance en base de données).

ABCDE est un patron composé de 4 couches principales et d'1 supplémentaire pour délimiter la présentation des acteurs. La matérialisation conceptuelle de la position des acteurs a pour objectif d'améliorer la compréhension, les données issues des acteurs passent successivement par les couches B, C et D pour être traitées dans la couche métier E. La proposition est agrémentée de partitions orthogonales [BCK03], délimitant les grandes fonctionnalités du système.

Le tableau 10.1 décrit les couches du patron ABCDE :

A: Actor - Le niveau représente les acteurs (au sens UML du terme) qui peuvent être soit des modèles boite-noire d'éléments délivrés par des fournisseurs externes (par exemple des radars), soit des systèmes connus (tels des simulateurs développés en interne).

B: Boundary - La couche marque la frontière du système et contient la description des règles d'interface, à l'exemple de la gestion des protocoles de communication.

C: Control - Le niveau a un rôle de coordinateur et d'adapteur (proche du PAC) et se concentre sur la gestion de la cohérence des données d'interface et leur translation en des données métiers.

D: Data - La partie concerne la gestion des données métiers, notamment leur persistance.

E: Entity - L'expertise métier est située dans la couche *E:Entity*, qui regroupe l'ensemble des règles de traitement métier et leurs dépendances.

Tableau 10.1: Le patron d'architecture ABCDE.

	Nom	Description	
E	Entity	Règles métiers de traitement	Expertise métier
D	Data	Données métiers / Persistance	
C	Control	Données d'interface	Interface
B	Boundary	Règles d'interface	
A	Actor	Equipements en regard	

Perpendiculairement à cette décomposition, l'architecture logique est structurée en partitions [BCK03] (ou tiers logiques), pour une découpe en fonctionnalités du système : différents traitements tactiques, simulation de l'environnement, etc. Les couches horizontales mettent en exergue des points de vue architecturaux génériques alors que les partitions verticales sont davantage corrélées au domaine métier.

Dériver une architecture depuis des patrons permet de mettre en avant les décisions prises, facilitant la modification de celle-ci lors de l'apparition de nouvelles exigences.

Description dynamique

L'ingénierie système, précédant l'ingénierie logicielle, suit le processus IDM (niveaux *Contexte*, *Logique*, *Physique*) et transmet, entre autres, une découpe en sous-systèmes de la conception de la solution. Cette première découpe forme les partitions initiales au niveau logiciel, qui peuvent être affinées par la suite. À partir de cas d'utilisation ("Use-Cases"), "Goal-Cases" [LL04] et de partitions l'ingénierie logicielle va alimenter le canevas fourni par le patron d'architecture, permettant le franchissement d'un fossé syntaxique et sémantique [GEEM03] : le passage du problème à celui de la solution (au niveau logique dans un premier temps).

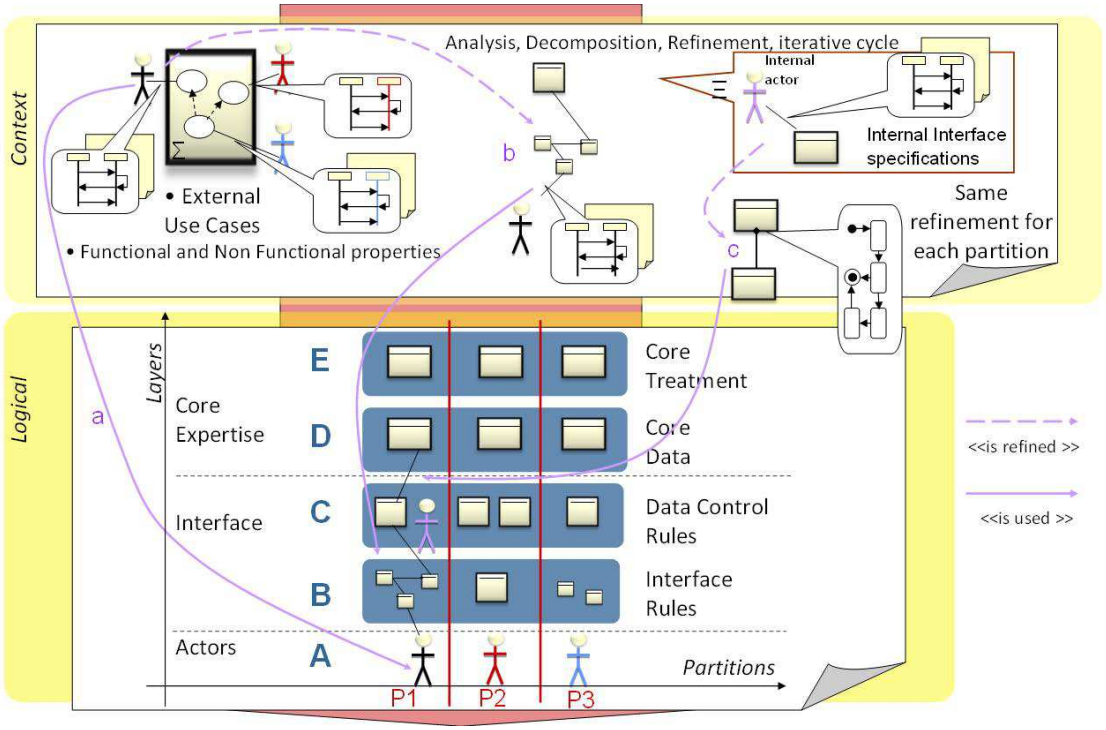


Figure 10.1: Dérivation de l'architecture depuis le patron.

La figure 10.1 donne un aperçu de la méthodologie de dérivation, afin d'illustrer l'utilisation du motif architectural. Au niveau *Contexte*, les entrées sont des "Use-Cases". Les "Use-Cases" définissent, avec une orientation intention et objectif, des ensembles d'interactions entre des acteurs externes au système étudié [Coc97, LL04]. Il est bien entendu important de structurer convenablement ces "Use-Cases" afin d'éviter toute ambigüité. Dans ce schéma, les partitions (par exemple P1, P2, P3 sur l'axe horizontal pour les traitements tactiques de corrélation de pistes, détection d'objets et de simulation) soulignent les fonctionnalités principales. Les "Use-Cases" externes (définissant le contour du système) sont dérivés pour constituer les couches d'interface (*Boundary* et *Control*) : les acteurs sont représentés au niveau *A* (cf. figure 10.1 - a) et les diagrammes de séquences sont traduits dans des composants comportementaux au niveau *Boundary*. Les principaux "Use-Cases" définissent les éléments de la couche *Control*. Des cycles itératifs de raffinement et décomposition (cf. figure 10.1 - b) sont appliqués pour remplir graduellement l'architecture logique, en partant de la couche *Boundary* vers les supérieures. Par exemple, les spécifications des interfaces internes définissent la relation entre les niveaux *C* et *D* (cf. figure 10.1 - c). Les capacités sont caractérisées, par ex. par le biais de diagrammes de "state-chart", et alimentent les niveaux domaine métier (*D* et *E*).

Au niveau *Logique*, les extrants se doivent d'être conformes au motif ABCDE, organisés en partitions (celles de l'ingénierie système ou un raffinement de celles-ci). Par la suite l'architecture obtenue sera projetée au niveau physique, avec une corrélation plus ou moins forte, les découpes logiques pouvant être raffinées par la plate-forme physique (cas d'applications distribuées par exemple). L'application créée par le patron architectural est basée sur un ensemble de plate-formes et "frameworks" existants. Par exemple, dans le cas d'applications distribuées, l'intergiciel peut être CORBA [OMG06] ou le "Data Distribution Service" (DDS) [OMG07] ; les composants

de l'intergiciel pouvant être définis par le modèle "CORBA Component Model" (CCM). La figure 10.2 illustre le positionnement du motif architectural ABCDE (avec deux partitions) par rapport aux couches physiques.

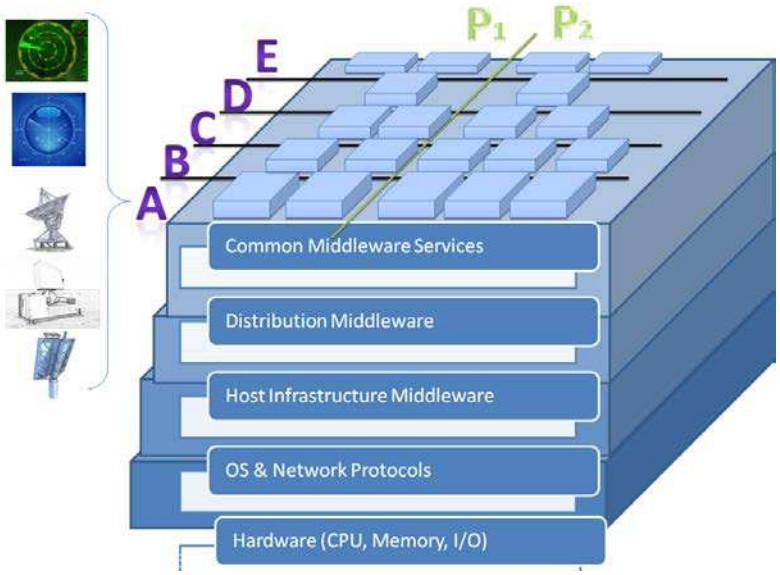


FIGURE 10.2: Couches d'intergiciel et patron ABCDE avec partitions.

10.4 Conséquences de l'application de ABCDE

Les conséquences de l'application sont abordées ici par une description des bénéfices de la mise en œuvre du patron et de ses limitations. Concernant les limitations, ces dernières peuvent provenir de la possibilité d'inadéquation entre le patron sélectionné et le type d'applications ciblées, ce qui n'est pas le cas dans notre cas d'application (cf. section 12.3).

De manière générale, il est à noter que les travaux existants (par ex. [BMR+96, Bos00]) évoquent de nombreux bénéfices pour les patrons en "couches", notamment en terme de facilité d'utilisation et de maintenance.

Lors de la création d'une architecture logicielle, il est essentiel d'identifier les points clés permettant à cette architecture de survivre à l'évolution, et mieux encore, d'évoluer proprement et harmonieusement. Afin de créer une architecture durable et pérenne, il est important d'être en capacité d'identifier un certain nombre de changements susceptibles de se produire.

Le patron d'architecture présenté dans la section ci-avant introduit un ensemble de conventions et règles de par ses couches et partitions, permettant d'organiser l'architecture par des modules cohésifs. Par conséquent, un changement des acteurs, source principale de variabilité de notre Ligne de Produits, ne devrait pas être propagé aux confinements du système *(Core expertise)*, sinon être contenu dans les couches d'interface, cf. schéma figure 10.3. La variabilité des acteurs *(A:Actor)* est pour une grande part, contrôlée par les couches *B:Boundary* et *C:Control*. Se conformer aux règles du patron d'architecture permet une amélioration de la modifiabilité, la réutilisabilité et l'extensibilité de la solution résultante.

Le motif d'architecture est un support pratique pour la prise de décision de conception, et par conséquent nécessaire à la résolution de la variabilité.

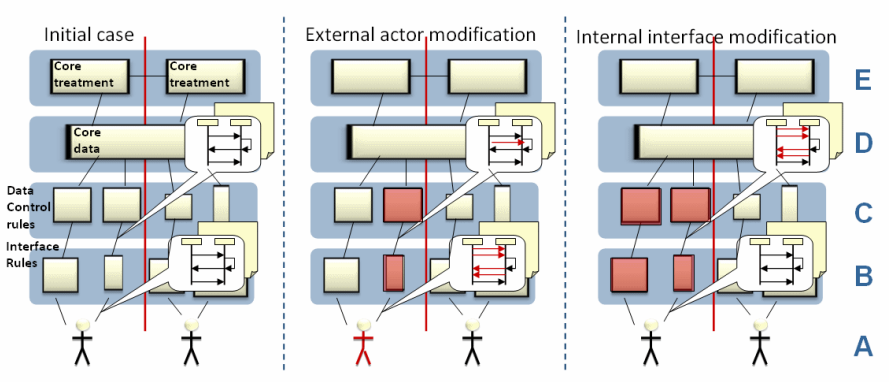


Figure 10.3: La propagation est résorbée par les couches d'interface.

Le patron d'architecture est partie prenante d'une stratégie de contrôle de la dérivation de produits et de la gestion de l'évolution de la *ligne de produits* (en facilitant la délimitation de son périmètre d'action - "scoping" [JV10] - de la *ligne de produits*). Le patron a une incidence sur la plate-forme et également sur l'ensemble de l'espace de modélisation, y compris sur la structure des modèles de variabilité[1]. L'explicitation des règles améliore la compréhension, facilite la maintenance et réduit le risque d'introduction de bugs.

De plus, l'application d'un tel patron, se focalisant sur le métier et les interfaces, organise la variabilité et la minimise. Il est toujours avantageux d'avoir un nombre de points de variation le plus faible possible, réduisant de ce fait les efforts de gestion et les possibilités d'incohérences.

10.5 La connexion au langage PLiMoS

La séparation des préoccupations actée par la constitution d'un motif de conception architectural est soutenue par une réflexion et peut être délimitée par des relations intentionnelles. À un niveau de granularité plus fin, des *relationships* permettent une matérialisation opérationnelle des contraintes liées au patron.

La mise en œuvre d'un motif d'architecture au travers du langage $PLiMoS_{specialisation}$ peut être réalisée, pour le moins, de deux manières distinctes, avec une explicitation ou non de la modélisation de la structure du motif.

À titre illustratif, prenons pour exemple un patron d'architecture en trois couches : *Presentation*, *Business*, et *Data Access*. La strate *Presentation* modélise l'interface avec l'utilisateur, celle dite *Business* fait référence à l'implémentation de la logique métier, et enfin la couche *Data Access* propose de fournir l'accès aux différentes sources de données. La figure 10.4 met en évidence cette décomposition.

Les relations intentionnelles sont données par les relations $\mu\#14$, P représente la modélisation de la strate *Presentation*, B celle du *Business*, et DA celle de *Data Access*. Les intentions de modélisation sont différentes et expriment un décalage dans les préoccupations adressées ; les modèles de couches supérieures expriment un contenu étendant celle de niveau strictement inférieur.

$$P \xrightarrow[\mu\gamma]{} B \qquad B \xrightarrow[\mu\gamma]{} DA \qquad \mu\#14$$

[1] expliqué plus en détails dans la section 12.3

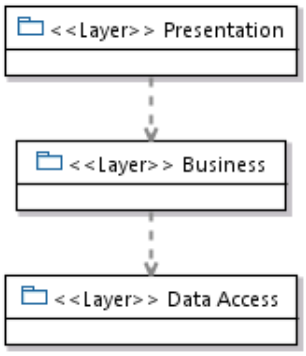


FIGURE 10.4: Exemple de patron d'architecture.

Les *relationships* les spécialisant prennent en considération l'explicitation ou non du modèle de représentation du patron (du type modèle de la figure 10.4).

10.5.1 Explicitation de la modélisation du motif architectural

Dans le cas d'une explicitation du modèle, le lien sémantique est créé entre l'entité réifiant un élément du patron (dans le cas illustratif présent il s'agit d'une couche) et les éléments s'apparentant à la préoccupation réifiée.

La relation considérée dans l'exemple est une relation d'implémentation de type 1-n, hétérogène, *inter-concern*, concret, avec aucun attribut ni aucune opération associé. Cette relation se formalise de la manière suivante : soit C l'ensemble des couches du motif architectural ; soit E l'ensemble des éléments de l'espace de modélisation (hors modèle du motif) ; notons RCE l'ensemble des relations couches/éléments ; et enfin considérons RCC l'ensemble des relations inter-strates. Les relations sont définies telles :

- $RCE \subseteq C \times \mathcal{P}(E)$
- $RCC \subseteq \mathcal{P}(C)$

Notons "$\rightarrow$" la relation de C dans E, les relations RCE doivent respecter les règles de bonne formation suivantes :

- tout élément appartenant à une relation ne peut appartenir à une autre :
  $\forall c1, c2 \in C \cdot \nexists e \in E | c1 \rightarrow e \wedge c2 \rightarrow e$;
- tout élément du motif doit être lié à un ou plusieurs éléments de E :
  $\forall c \in C \cdot \exists e \in E | c \rightarrow e$;

Les éléments liés pouvant être hétérogènes et à divers niveaux d'abstraction, ce type de relation est complexifiable afin d'y ajouter de la sémantique.

La figure 10.5 propose une visualisation du patron de la figure 10.4 et de son implémentation au travers de composants. Dans cette représentation, il est choisi de faire apparaître en contenu des strates les éléments d'implémentation, ainsi que différentes relations de dépendances (en pratique visualisables ou filtrables à la demande à l'aide des filtres Obeo Designer, base de l'implémentation de l'outillage PLiMoS, plus de détails au chapitre 13).

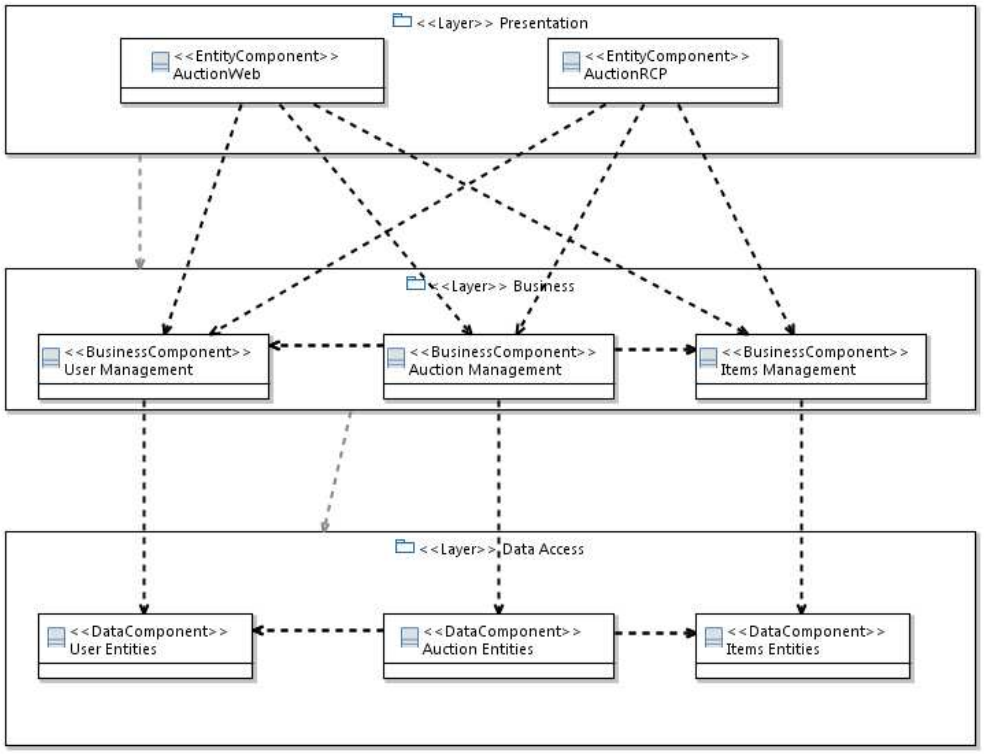


Figure 10.5: Exemple de patron d'architecture et une première visualisation de sa réalisation.

10.5.2 Absence d'explicitation de la modélisation du motif architectural

Dans le cas d'une non explicitation du modèle, la relation symbolise le concept du motif architectural, et la relation devient n-aire, avec toujours la possibilité de mettre en valeur certains éléments de modélisation ou "type" de relations. La figure 10.6 présente le cas illustratif sans modélisation du patron, les choix de conception de l'outillage graphique s'étant portés sur une représentation des *relationships* sous une forme de nœuds (afin de représenter des relations d'ordre supérieur à deux), connectés entre eux et avec les éléments de modélisation. Dans le cas présent, la relation est considérée de type horizontal, homogène, *inter-concern*, concret et sans attribut ni opération. Ce type de représentation donne lieu à des graphes labélisés.

La modélisation des *relationships* peut se formaliser par le 3-uplet suivant $\langle E, RE, RRE \rangle$, où sont distingués :

- E, l'ensemble des éléments de l'espace de modélisation ;
- $RE \subseteq \mathcal{P}(\mathcal{P}(E))$, l'ensemble des *relationships* liant les éléments entre eux ;
- $RRE \subseteq \mathcal{P}(RE)$, qui représente les relations entre *relationships* (considérée comme 1..1, homogène, verticale, sans attributs ni opérations).

Les conditions de bonne formation sont :

- une et unique *relationship* par concept du patron : $card(RE) = n \in \mathbb{N}$, avec n nombre de concepts distincts du motif architectural ;
- un élément e est associé une et une seule fois :
  $\forall e1, e2 \in E | e1 \rightarrow e2 \cdot \nexists e3 \in E | e3 \rightarrow e1 \vee e3 \rightarrow e2$.

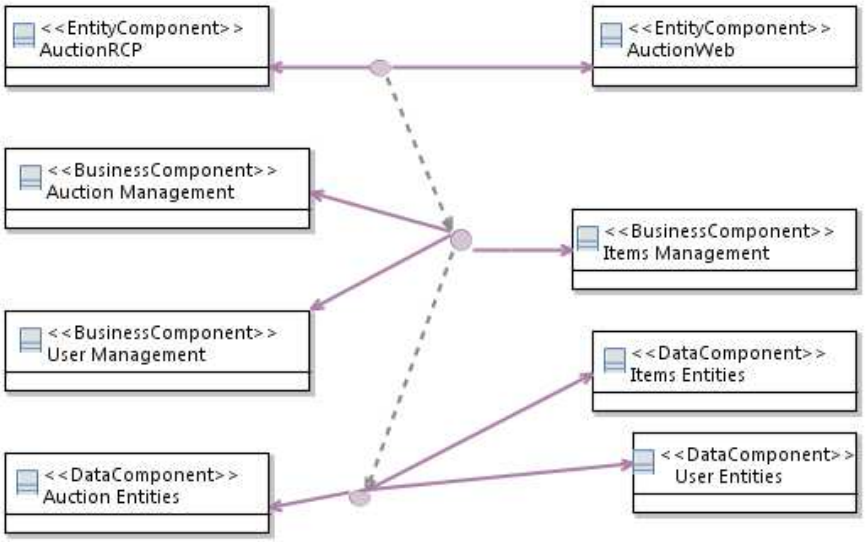


FIGURE 10.6: Exemple de patron d'architecture et une seconde visualisation de sa réalisation.

Cette mise en relation des divers concepts d'architecture au travers du langage PLiMoS permet d'envisager une mise en application relativement aisée de contraintes de conception, qui déterminent le degré d'évolution possible de l'architecture dans le temps. Le langage offre ainsi une possibilité de vérification de conformité à un standard architectural.

10.6 En résumé

Une architecture de *ligne de produits* ouverte est garante d'un meilleur déroulement de l'évolution. L'obtention de celle-ci est favorisée par la sélection et l'application d'un motif architectural adapté au domaine métier et au type de système ciblé. Il s'agit de réduire l'arbitraire de la description en mettant en évidence dans les agrégations de formes, celles qui sont nécessaires ou probables. Dans la pratique, le patron *ABCDE* propose une structuration en couches et partitions répondant aux besoins applicatifs chez Thales Air Systems. Il théorise la dissociation de l'information architecturale et de l'espace de modélisation des *lignes de produits*, tout en minimisant la variabilité et sa gestion. Le patron d'architecture se traduit dans le langage PLiMoS de modélisation de l'espace de modélisation, et est partie prenante de la stratégie de contrôle de la gestion de l'évolution de la *ligne de produits*.

CHAPITRE 11

Une évolution par extension

*"*Tant *de faits divers* [...] *en faveur de l'*évolution*, prouvent assez que les Corps organisés ne sont point proprement* engendrés *; mais qu'ils préexistoient* originairement *en petit."*

—Charles Bonnet, *Considérations sur les corps organisés, 1762*

Un autre volet de la thèse part du constat qu'il est difficile, voire impossible de définir et de faire évoluer une *ligne de produits* proactivement, c.-à-d. une LdP qui adresserait l'ensemble des préoccupations de la famille en une seule étape, ou dont les évolutions seraient planifiées avec une anticipation des futures exigences. L'idéal est dès lors de proposer une démarche de construction incrémentale où les ingénieurs peuvent en plusieurs étapes, et avec des facilités d'adoption, ajouter de nouvelles préoccupations à leur architecture et ainsi augmenter le périmètre de la *ligne de produits*.

Le chapitre précédent s'attelait à définir un environnement architectural coercitif mais ouvert sur une évolution de variabilité ciblée, le présent chapitre concerne l'aspect processus d'évolution, toujours dans le cadre de modélisation défini par PLiMoS. L'approche proposée tend à resserrer les liens entre l'ingénierie de l'application et celle du domaine, et par là aborder une évolution incrémentale et réactive de la *ligne de produits*.

Ce chapitre rappelle brièvement la problématique du manque de réactivité des approches existantes (section 11.1) pour caractériser l'évolution par extension, cible de l'approche (section 11.3). Le processus d'évolution incrémentale de la *ligne de produits* est exposé dans sa globalité (section 11.4), puis plus finement décrit concernant des étapes critiques (section 11.5). Enfin, la dernière section met en perspective le langage PLiMoS et l'évolution de la Ligne de Produits (section 11.6).

Certains éléments de ce chapitre figurent dans les travaux présentés à MAPLE 2012 [CCJM12a].

11.1 D'une réactivité requise

La *ligne de produits* doit réagir dans un temps le plus réduit possible et lisser au maximum la courbe des efforts à fournir. Le besoin d'évolution peut affecter n'importe quelle ingénierie ou

quel niveau d'abstraction de modélisation, et en affecte généralement plusieurs.

L'objectif est d'atteindre une certaine agilité dans la démarche d'évolution de la *ligne de produits*. L'agilité recherchée signifie, dans le contexte, une habileté à l'adaptation aux situations mouvantes dans un temps le plus court, d'une manière appropriée et effective, elle ne fait pas référence au courant de développement du même nom.

L'agilité fait appel à une stratégie de transition incrémentale. Incrémentale dans l'effort à produire, et incrémentale dans le retour sur investissement. Une évolution incrémentale est relativement pragmatique et tend à fournir des bénéfices suffisants dès l'incrément suivant. Ce faisant, le développement courant se réalise sur des efforts passés, et idéalement la transition paye son propre développement.

En outre, dans les approches existantes de *lignes de produits*, un manque de flexibilité de la dérivation de produits est notable, qui ne permet pas d'absorber les exigences non-anticipées. Les développements spécifiques réalisés ne sont généralement pas réintégrés, et encore moins de manière automatique, dans la *ligne de produits*. Un automatisme complet de ce processus de remontée des artefacts de l'ingénierie de l'application vers celle du domaine n'est, d'une part pas aisée, mais également pas souhaitable : ce qui relève des choix de conception d'une *ligne de produits* doit être laissé à la charge d'un expert métier.

11.2 Une réactivité fondée sur l'ingénierie de l'application

Le management de produits dans les *lignes de produits* comprend son positionnement (périmètre d'action) dans une perspective d'interaction entre la vision marketing de la LdP (des produits ciblant un marché particulier) et l'aspect technique (regroupé autour d'une plate-forme commune) traité dans les activités d'analyse du domaine ("*Domain Analysis*") et de conception ("*Domain Design*"), pour rappel cf. fig. 4.1, page 62. Des équipes distinctes sont généralement en charge de l'adaptation et de la spécialisation des nouveaux produits à partir des *core assets* mis en partage.

La *ligne de produits* au centre des débats montre un besoin de réactivité d'évolution, induite par sa nature et son périmètre : il s'agit de gérer en LdPs des produits d'une même gamme, créés à différents instants, suivant des besoins clients émergents. Le phénomène d'évolution peut apparaitre suite à l'émergence d'exigences spécifiques produit, de requêtes de modification, ou de signalement de bug lors du processus de développement du produit.

En outre, plusieurs projets produit (ou affaires) peuvent se retrouver développés en parallèle, c.-à-d. de manière concurrente. Les aspects collaboratifs, bien qu'extrêmement importants, sont posés comme des travaux perspectifs, les préoccupations actuelles se situant sur la gestion de la dérivation et la remontée rapide des évolutions requises.

Dans ce contexte, la LdP devrait tirer les bénéfices des retours d'expériences et de conception réalisés dans la phase de spécialisation de l'ingénierie de l'application et les partager en les introduisant dans sa base pour une réutilisation dans les réalisations à venir.

Une vue d'ensemble de l'approche est fournie par la figure 11.1, qui présente une boucle rétroactive de développement pour l'intégration dans la *ligne de produits*. La phase d'ingénierie de l'application s'initialise par l'arrivée d'un nouveau besoin client déclenchant la dérivation de produit. Les nouvelles exigences peuvent nécessiter un développement spécifique, celui-ci ayant un impact possible à divers niveaux du processus de développement (de l'ingénierie système au matériel, en passant par divers niveaux d'abstraction - représentés ici par les niveaux de la méthodologie ARCADIA). Ces développements spécifiques sont réalisés par extension de produits, d'artefacts de modélisation existants, d'adaptations diverses et variées ; ces développements ne suivent pas forcément une méthodologie descendante, et de la rétroconception peut être nécessaire pour obtenir une complétude des modèles du produit. Par la suite se pose la question de la réutilisation de ces développements et donc de la réintégration de ces derniers dans la *ligne de*

produits (*Feedback loop*), cette remontée pouvant être en partie automatisée dans le cadre d'une augmentation par extension de la *ligne de produits*, ou nécessiter une réingénierie plus profonde *(Reingeneering Management & Development)*, qui ne fait pas l'objet de cette étude.

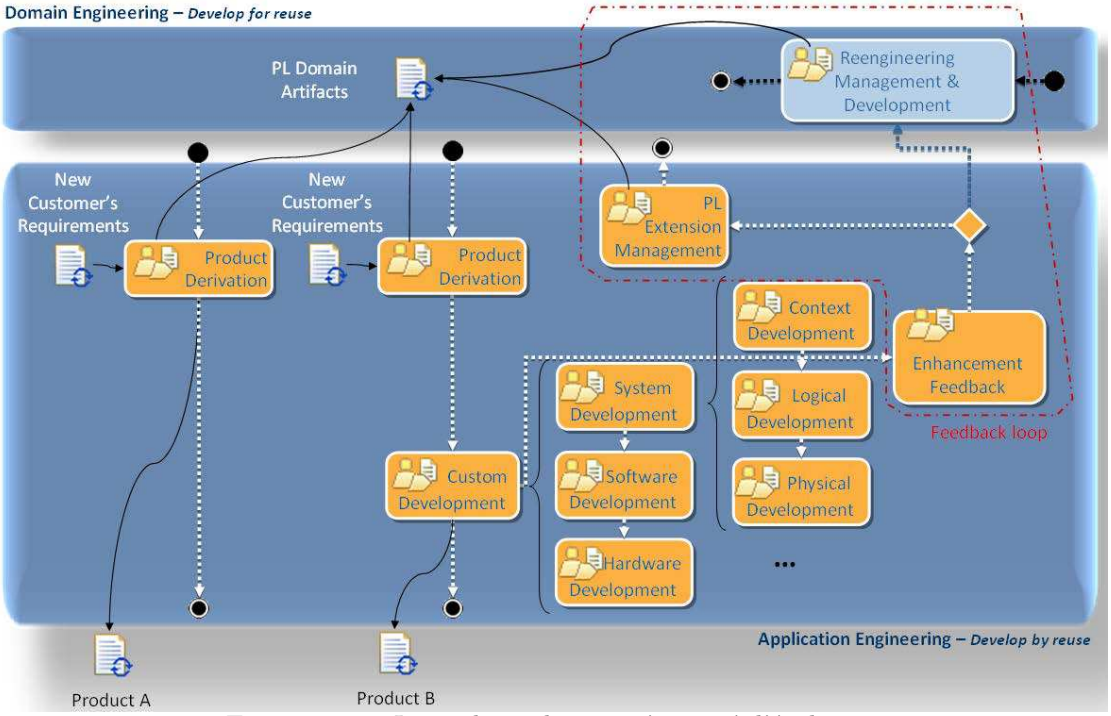


FIGURE 11.1: *Ligne de produits* et réactivité d'évolution

Porté par le besoin d'une approche réactive permettant l'intégration croissante de produits, la section suivante vise à définir l'évolution par extension de la *ligne de produits*.

11.3 Caractérisation de l'évolution par extension

Afin de caractériser l'évolution ciblée, dite par extension, cette section présente un ensemble d'opérateurs de modification d'artefacts des Lignes de Produits (ajout, modification, suppression), en commençant par s'interroger sur l'étape d'occurrence de cette action (c.-à-d. l'Ingénierie de l'Application ou du Domaine), pour déterminer le sous ensemble de ceux-ci qui s'apparente à de l'évolution par extension.

Ne sont considérés ici qu'un nombre limité d'opérateurs logiques de variabilité, par la suite identifiés par *And*, *Or* et *Xor*. Une commonalité de la LdP signifiée par l'opérateur *And* marque une obligation de sélection de variants. La notion d'optionalité d'un variant est également prise en considération. En outre, les artefacts constituant la LdP sont considérés comme étant de trois types : le modèle de variability *(VM)*, les *Core Assets (CA)*, et les liens les unissant (*mapping*).

Le tableau de la figure 11.2 présente un résumé des opérateurs d'évolution des produits et les classe en catégories suivant divers critères. De manière générale, la phase d'initiation peut appartenir soit à l'ingénierie de l'application, soit à celle du domaine. Chaque opérateur :

- a un type ;
- a un nom et une description ;

- a un impact sur les artefacts de l'espace de modélisation de la Ligne de Produits ;
- peut être classé en type de variabilité (la variabilité est considérée comme étant de "l'espace" si la modification impacte la structure du modèle de variabilité et "du temps" si la modification transforme des artefacts existants) ; une variabilité de l'espace est qualifiée de interespace de par son impact sur les différents espaces (principalement le modèle de variabilité, mais également dans une moindre mesure le "mapping" et les *core assets*), une variabilité dans le temps peut être de nature intra ou inter espace ;
- peut être traité dans une phase donnée (c.-à-d. de manière réactive par l'ingénierie de l'application, ou proactivement dans l'ingénierie du domaine).

Les modifications prises en considération par l'évolution par extension sont abordées sous l'angle de la préservation de la sémantique ou du comportement des modèles impliqués. Ces modifications apportent une extension (sauf cas explicitement listé ci-après) non destructive de la modélisation de la *ligne de produits*.

Les évolutions considérées sont les suivantes :

- Ajout d'un élément spécifique produit *(A1)* : l'opération concerne uniquement le produit et est considérée comme une option de ce dernier, impactant les trois types de modèles ;
- Ajout d'un nouveau variant *(A2)* : l'opération correspond soit à (i) l'ajout d'un variant dans un point de variation existant (de type *Or* ou *Xor*), ou à (ii) la création d'un nouveau point de variation. L'impact porte sur les trois types de modèles ;
- Ajout d'une nouvelle commonalité *(A3)* : l'opération cible donc une modification d'un point de variation de la LdP, avec un impact sur les trois types de modèles ;
- Modification de la commonalité *(M1)* : l'opération est relative à un changement de réalisation d'un point de variation existant et a un impact sur les *core assets* et le *mapping* ;
- Modification d'un variant *(M2)* : l'opération est relative à un changement de réalisation d'un variant existant, et a un impact sur les *core assets* et le *mapping* ;
- Modification des contraintes *(M3)* : l'opération est relative à une modification des contraintes (relaxe ou restriction), qui peut être relativement simple (du type ajout ou suppression de contraintes d'implication ou d'exclusion dans le modèle de variabilité), ou plus complexe impliquant une modification de la structure des points de variation et potentiellement des *core assets* et *mapping* ;
- Suppression d'un élément spécifique *(D1)*, d'un variant *(D2)* ou d'un élément commun *(D3)* : les opérations réduisent le périmètre de la *ligne de produits* et ont un impact global.

Les modifications concernant une granularité supérieure de manipulation de *lignes de produits* (ajout, fusion, séparation) ne sont pas abordées dans le tableau de la figure 11.2.

Le processus réactif d'évolution semi-automatique est limité à l'ajout d'éléments variables par des opérateurs logiques de type *Or* (ou création de *Xor*) - *A1* et *A2*. Cette extension de la *ligne de produits* n'a, en effet, aucun impact sur les produits précédemment créés.

Notons que l'opérateur *M1* peut être considéré comme partie prenante de l'évolution par extension si la relaxe de contraintes est admise, par ex. un opérateur *And* peut devenir un *Xor* (signifiant par là la reproductibilité des produits précédemment issus de la LdP). L'impact doit être vérifié en prenant en compte les contraintes de configuration transverses (exclusion mutuelle et nécessité), de même que les contraintes exprimées en logique propositionnelle. De manière similaire, l'opérateur *M3* est recevable dans le cas d'une expression de relaxe des contraintes.

11.4 L'évolution incrémentale par extension d'une LdP

L'évolution incrémentale par extension caractérisée par les opérateurs dans la section précédente promeut une exploitation systématique des efforts de conception. Ce processus est décrit dans cette section, et illustré par un cas d'application simple.

Starting	Type	Name & description	Impacted artifacts	Variability	Management in
AE & DE	Add	A1: Add product specific (option)	All	space	AE/DE
		A2: Add new variability (Or, Xor)	All	space	
		A3: Add new commonality (And)	All	Space & time	DE
	Modify	M1: Modify commonality (base model or VM)	Assets & mapping	time	
		M2: modify a variant (or existing option)	Assets & mapping	time	
DE		M3: modify constraints	All	Space & time	
	Delete	D1: delete product specific	All	Space & time	
		D2: delete variability	All	Space & time	
		D3: delete commonality	All	Space & time	
	Adding, Merging, Splitting Product lines…				

Extending scope

improvement

FIGURE 11.2: Résumé des opérateurs d'évolution de produits

11.4.1 Une exploitation systématique des efforts de conception

L'approche présentée ici promeut la découverte et l'exploitation systématique de nouveaux *variants* (et *core assets*) par la *ligne de produits*. Ce processus sur mesure introduit fig. 11.3 fournit soit (i) un retour semi-automatique pour l'intégration des éléments de produit (activité de récupération et incorporation des éléments de la *ligne de produits*, "*PL element Recovery & Incorporation*" de la figure), ou (ii) des retours d'information aux analystes et experts du domaine pour modifier ou étendre la modélisation du domaine pour les développements futurs (dans l'activité de restructuration du domaine, "*Domain Refactoring*" de la figure).

Ces retours peuvent également être bénéfiques à l'amélioration du processus d'analyse du domaine, en découvrant des manques possibles dans les méthodes originelles. Ceci est classé comme de l'évolution proactive et n'est conséquemment pas une activité de la dérivation de produits, mais bien une activité de l'ingénierie du domaine. Ce type d'évolution est plus complexe de par ses impacts sur les membres existants de la LdP et sur son périmètre. Il implique une plus grande expertise et une compréhension d'ensemble de la Ligne de Produits.

11.4.2 Introduction du cas d'exemple illustratif

Afin d'illustrer les propos, la section reprend l'exemple "*ItemCatalog SPL*" introduit précédemment (cf. section 3.2), et illustré pour rappel dans la figure 11.4.

La représentation de la variabilité est basée sur l'utilisation d'un modèle de *feature* et de *Core Assets* défini par un DSML. Dans cet exemple d'illustration, un sous-ensemble des diagrammes de classe UML exprimé en ECore est utilisé. Par conséquent, dans l'exemple, un *Core Assets CA* est défini tel un 8-uplet d'ensembles ⟨*EClasses, EReferences, EOperations, EAttributes, EEnums,*

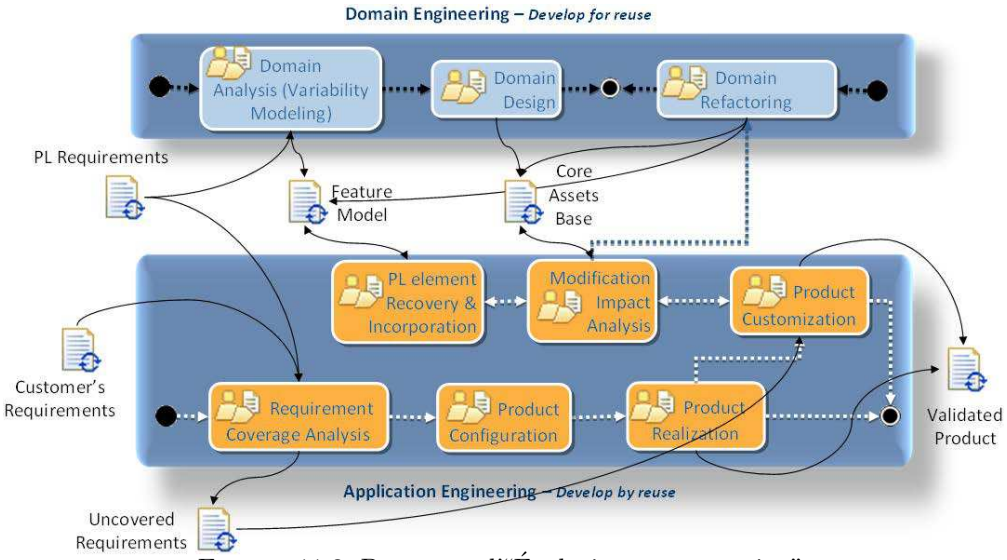


Figure 11.3: Processus d'"Évolution par extension"

EEnumLiterals, EPrimitiveTypes, EAnnotations⟩.

Toujours sur le même cas d'illustration, la figure 11.5 fournit un extrait des *core assets* correspondant à certains *features* de la figure 11.4 : *CatalogStructure*, *ProductInformation*, *3DImage*, *Categories* et *Description*.

Les *features* sont liés aux fragments de modèles ECore. Dans cette section, pour des raisons de compréhension, les relations établies dans la partie II ne sont pas mises en œuvre. Les *features* sont *réalisés par* des fragments de modèles (ensemble de *core assets*), et nous considérons $RR \subseteq C \times CA$ comme étant l'ensemble des relations de réalisation (*Realization* P). Un *feature* donné peut être réalisé par de multiples éléments ECore, nonobstant aucune relation n-aire n'est définie, le concept de *core assets* est réifié pour représenter un ensemble d'éléments.

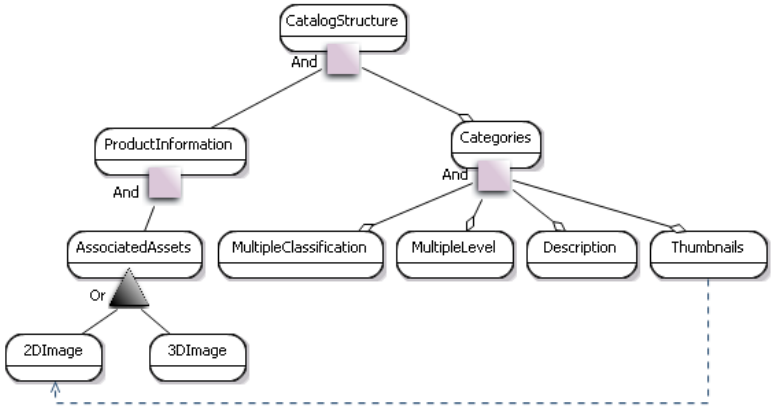


Figure 11.4: Exemple *ItemCatalog SPL* - modèle de *features*

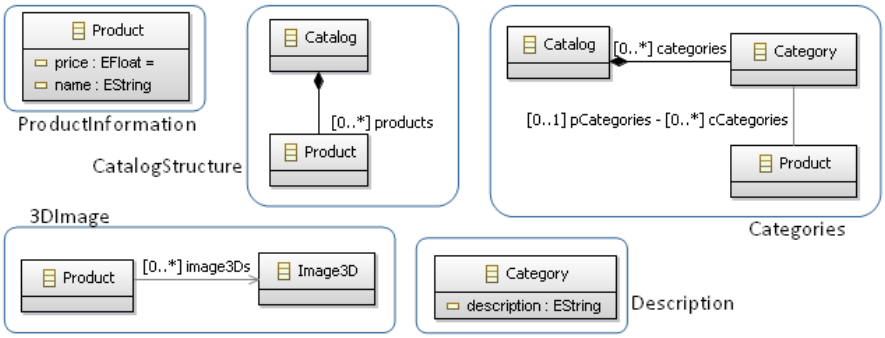


FIGURE 11.5: Exemple *ItemCatalog SPL* - des *core assets*

11.4.3 Un processus d'évolution par extension initié dans l'Ingénierie de l'Application

Les développements spécifiques et les efforts associés sont réalisés par des équipes lors de la dérivation de produits. Afin de faciliter la remontée des informations et de capitaliser sur ces efforts, un processus d'évolution semi-automatisé est introduit, prenant racine dans l'ingénierie de l'application. Par conséquent, la figure 11.3 décrivant le processus ne s'étend pas sur la description des activités de l'ingénierie du domaine.

Du côté Ingénierie de l'Application, en commençant par les exigences issues des besoins client, le processus suit les activités suivantes :

Analyse de la couverture des exigences : ("*Requirement Coverage Analysis*") les exigences sont évaluées pour séparer celles couvertes par la *ligne de produits* existante et celles qui ne le sont pas (si la LdP couvre la totalité des besoins du client, l'ensemble des exigences non couvertes est bien évidemment vide). L'approche actuelle considère un processus d'évaluation manuelle.

Configuration du produit : ("*Product configuration*") partant des exigences couvertes par la LdP, une sélection des *features* requises du FM est réalisée. La configuration peut être partielle, si les *features* existants ne couvrent pas tous les besoins. La configuration est par la suite validée et doit être conforme aux contraintes d'exclusion mutuelle du FM. Cette configuration fait office d'intrant pour la phase de réalisation du produit. Cette configuration de "base" est une configuration partielle formant l'ossature du produit à réaliser.

La réalisation du produit: ("*Product realization*") l'étape consiste en l'assemblage d'un sous-ensemble des *core assets* partagés par la *ligne de produits*, en cohérence avec la configuration. Dans le cas d'une configuration non partielle, un produit valide est obtenu à la fin de cette activité. La méthode de dérivation supporte un processus incrémental de dérivation, permettant l'ajout pas à pas de *core assets* au produit résultant.

Personnalisation du produit: ("*Product Customization*") cette phase correspond à l'exécution de développements spécifiques au produit par les ingénieurs de l'application. Ceux-ci peuvent être manuels ou résulter de transformations de modèles par exemple. Toujours en suivant notre exemple illustratif, la figure 11.7 présente le produit obtenu avec la configuration $c_{example}$ et une modification permettant de supporter un stockage de vidéos. Il est important de noter la différence entre des modifications et des améliorations effectuées dans l'environnement « contrôlé » de la *ligne de produits* et en dehors. La figure 11.6 présente trois types d'évolutions possibles ; le premier cas facilite l'analyse d'impact des modifications (les modifications sont progressivement absorbées par la LdP, comme le per-

met la boucle dans le processus entre l'activité de personnalisation du produit - "*Product Customization*" - et celle de l'analyse d'impact - "*Modification Impact Analysis*" - dans la figure 11.3). Les modifications sont plus facilement détectées et ciblées, les liens entre les *features* et les *core assets* sont préservés au maximum, etc.. Plus le fossé est important entre le produit nouvellement modifié et la plate-forme d'artefacts connus de la LdP, plus difficile sera l'analyse à réaliser, avec par conséquent un risque accru d'imprécisions. La chronologie des cas de la figure 11.6 présente une gestion des modifications de plus en plus décorrélée de la LdP, se terminant par une nécessité de reconstruction complète des liens avec les *core assets*.

Analyse d'impact des modifications : ("*Modification Impact Analysis*") les modifications ont des niveaux différents de criticité qui peuvent mener à une restructuration du modèle du domaine ou à une évolution par extension soutenue par une aide à la prise de décisions. Pour rappel, le tableau de la figure 11.2 résume les opérateurs d'évolution par extension. En fonction des résultats de l'analyse, l'activité suivante est soit une restructuration du domaine[1], ou une étape de récupération et d'incorporation des nouveaux éléments dans la base ("*PL element Recovery & Incorporation*"). Cette activité concerne la recherche de similarité entre les éléments du nouveau produit et les *core assets* existants, ainsi que la suggestion d'intégration dans la LdP (dans les modèles de variabilité et dans la base des *core assets* ; elle est décrite plus en détails ci-après.

Récupération et incorporation d'éléments : ("*PL element Recovery & Incorporation*") en prenant en considération les résultats de l'impact sur le modèle de variabilité donné par la phase "*Modification Impact Analysis*", cette activité vise à réaliser les modifications dans le modèle de variabilité et à incorporer les nouveaux *core assets* dans la base. Brièvement, les *core assets* délimités dans l'étape précédente sont (i) validés par l'utilisateur (la validation repose sur l'expertise des ingénieurs) et (ii) adaptés à la base de *core assets*.

Dans la figure 11.8, la première partie présente la modification du FM résultante (ajout du variant *Video*, relié à ses parents par un opérateur *Or*), la deuxième partie décrit le nouveau *core asset* obtenu.

11.5 Le processus détaillé de découverte des *core assets* et la révélation de la variabilité

L'activité d'analyse d'impact de la modification ("*Modification Impact Analysis*") est raffinée par le processus présenté par la figure 11.9, pour (i) vérifier si des *core assets* existants dans la LdP peuvent être identifiés, (ii) établir le périmètre des nouveaux *core assets*, et (iii) identifier le type de variabilité impacté et, après validation, autoriser l'évolution par extension.

Soit *product* le produit nouvellement modifié et à analyser, ce modèle étant défini tel un ensemble d'objets. Ensuite, considérons que $CAset$ est l'ensemble de tous les *Core Assets* de la LdP (contenus dans la base). Enfin, considérons maintenant le morphisme g permettant l'identification des objets de *product* qui sont définis dans $CAset$, tel que g est un morphisme bijectif de *product* dans $CAset$. L'identification des éléments similaires est réalisée par une technique basée sur des signatures [RFG+05] (les détails sont donnés dans la section suivante).

L'ensemble R_{PL} représente l'ensemble des objets du produit qui sont définis comme des *core assets* existants dans la *ligne de produits*. Un nouvel objet *core assets obj* de *product* est dans R_{PL} s'il existe un objet *core assets obj'* dans $CAset$ tel que $g(obj) = obj'$, c.-à-d. :

$$R_{PL} = \{obj \in product \mid \exists obj' \in CAset, g(obj) = obj'\} .$$

[1] à noter que les modifications peuvent très bien rester spécifiques au produit et ne pas réintégrer la LdP

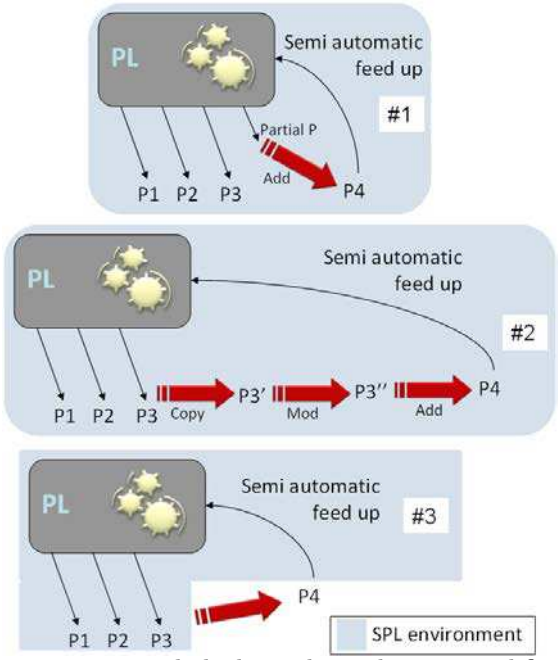


FIGURE 11.6: Environnement de la *ligne de produits* et modifications évolutives

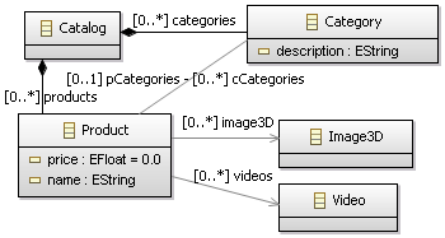


FIGURE 11.7: L'exemple du *ItemCatalog SPL* - produit étendu avec des capacités de stockage vidéo

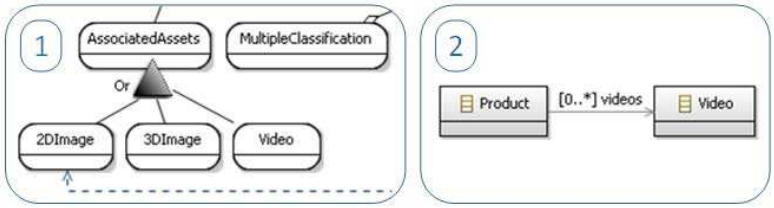


FIGURE 11.8: L'exemple du *ItemCatalog SPL* - modification du FM et découverte des *core assets*

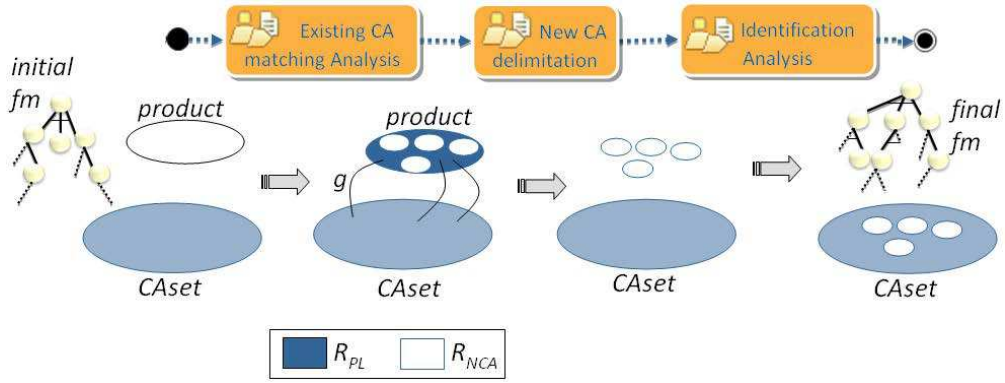


Figure 11.9: Analyse de l'impact des modifications

De manière similaire, R_{NCA} représente l'ensemble des nouveaux *core assets* dans le produit :

$$R_{NCA} = \{obj \in product \mid \nexists obj' \in CAset, g(obj) = obj'\} .$$

En partant de R_{NCA}, des sous-ensembles joints sont identifiés et séparés du modèle produit ; chaque nouveau sous-ensemble NCA_i ($i \in \{1, .., n\}$, n étant le nombre de sous-ensembles identifiés) représente un nouveau *core assets* potentiel et est analysé pour déterminer une identification possible de variabilité (les détails sont fournis en section 11.5.2).

11.5.1 Identification de *Core Assets* existant

L'approche utilise pour la composition des *Core Assets* une technique orientée modélisation par aspects nommée Kompose [FBFG07]. Elle utilise l'introspection et est basée sur le principe d'une fusion systématique d'éléments identifiés comme identiques. Dans cet outil, l'identification des éléments similaires est effectuée par comparaison de leurs signatures [RFG+05], et pour chaque élément fusionnable, l'utilisateur peut en personnaliser la signature. Par exemple, la signature d'une méthode peut être son nom, ou son nom et les types de ses paramètres.

En conséquence, pour identifier des nouveaux *core assets* à partir d'un produit donné, cette même technique d'identification est employée, avec les mêmes définitions de signature. Afin d'identifier les *core assets* existants, l'algorithme réalise un parcours de la base pour comparer chaque *core assets* de la LdP avec le produit. Les *core assets* similaires sont identifiés s'ils correspondent à une même signature. De plus, certaines signatures sont définies à partir d'identifiants ; pour chaque type d'élément est défini un dictionnaire des synonymes pour étendre la similarité des signatures avec une similarité attributionnelle. Par exemple, avec l'hypothèse que la signature d'une classe est définie par son nom, le dictionnaire peut stocker pour ce type d'élément (la classe) : le nom "*Image2D*" est dans ce cas synonyme de "*Img2D*". Quand deux mots ont un haut degré de similarité attributionnelle, ils sont dit synonymes. La détection et la conservation des relations d'Hyponymie/Hyperonymie n'est pas réalisée, mais fait partie des perspectives, car elles pourraient présenter des bénéfices, notamment dans l'optique d'une comparaison de diagrammes de classes utilisant la notion d'héritage.

En conclusion, pour situer le travail par rapport aux travaux relatifs à la comparaison de modèles (travaux qui n'apparaissent pas dans l'état de l'art du document), de nombreuses techniques de comparaison existent, s'appliquant sur des fichiers textuels, ou des documents structurés (par ex. XML). De plus, la discipline holistique de gestion de configuration propose de nombreux outils, techniques et processus de détection et changement des artefacts logiciel. D'autre part,

dans la communauté IDM, des recherches ont été réalisées avec des objectifs de gestion de version, de différenciation, de comparaison et de fusion de modèles. Les approches sont basées sur la comparaison (i) d'identifiants (par ex. [AP03]), (ii) de signatures (par ex. [RFG+05]), (iii) de métriques (par ex. EMF Compare [3])), et ciblent notamment UML. Cependant, il ne ressort pas de la littérature des approches ciblées sur les modèles et les *lignes de produits* alors que la présente approche traite ce cas, avec comme souci pratique de réutiliser la méthode de composition positive des *assets*.

11.5.2 Processus de remontée des *core assets*

L'algorithme suit la logique exposée dans la figure 11.10. Dans celle-ci, les questions correspondent à des étapes de communication avec l'utilisateur, implémentées au travers de fenêtres d'assistant sous *eclipse* ("*wizards*"). En effet, le processus d'incorporation des *core assets* est semi-automatique et entraine une proposition de solutions à l'utilisateur pour validation. L'approche repose sur l'expertise métier des utilisateurs et un retour complètement automatique n'est ni concevable, ni désirable. Par exemple, une différence sur le nommage de deux éléments mène à la question : "*For the X type of element, are A and B considered to be synonymous ?*"[2]. Les utilisateurs ont, suivant les cas, des choix d'action multiples (*choice ?* dans la fig. 11.10).

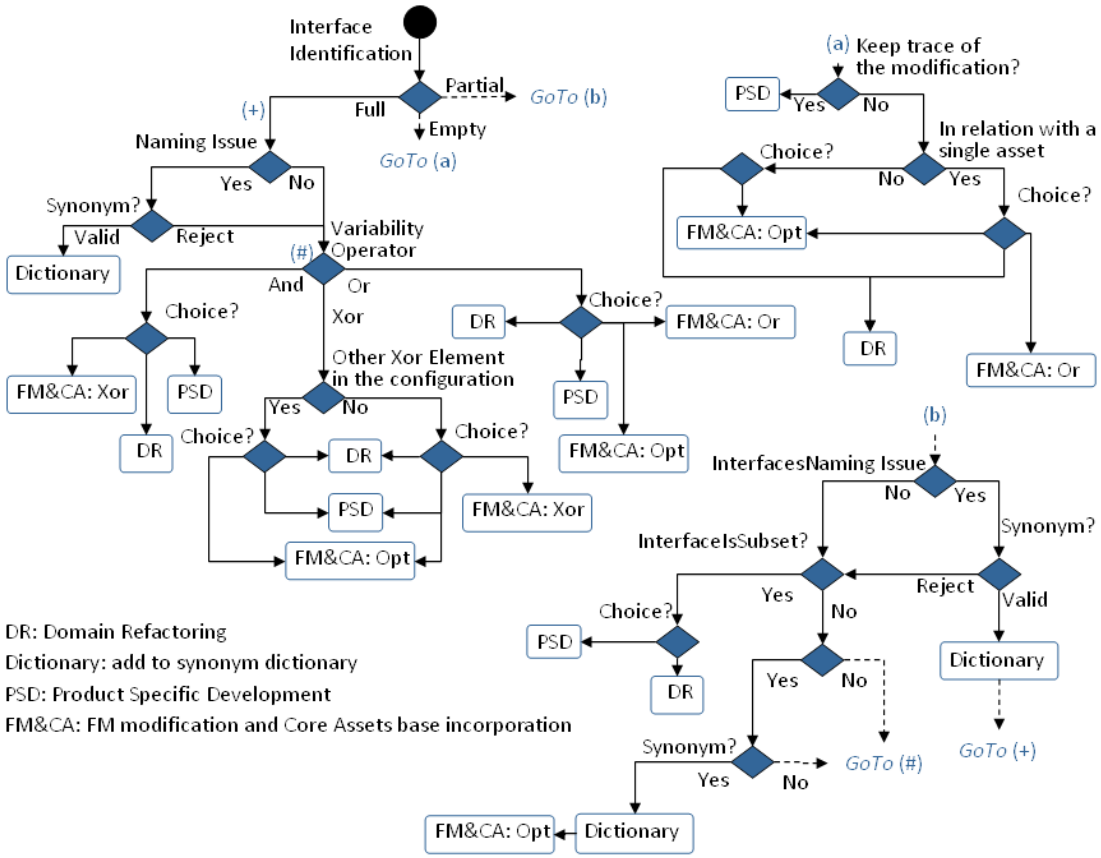


FIGURE 11.10: Logique de l'algorithme de retour

[2]Pour le type d'élément X, A et B sont ils considérés comme synonymes ?

Recherche de similarité d'interface

En partant des sous-ensembles joints de R_{NCA}, pour chaque NCA, l'étape d'identification des interfaces *(Interface Identification)* est exécutée. Cette dernière vérifie si le fragment de modèle NCA possède la même interface qu'un *Core Assets* existant CA.

L'interface d'un fragment de modèle NCA est composée de ses éléments en relation avec d'autres éléments de modèle. Soit I_{NCA}, l'ensemble des éléments d'interface d'un fragment de modèle ($I_{NCA} \subset NCA$). Dans la perspective d'une approche de fusion par identification de signatures, les interfaces sont contenues dans les *assets*. Dans le cas des diagrammes de classes par exemple, les relations entre les classes définissent son interface, les références et généralisations.

Considérons g' comme étant le morphisme permettant l'identification des éléments de NCA qui sont définis dans un CA (le morphisme bijectif de NCA dans CA), c.-à-d. :

$$I_{NCA} = \{element \in NCA \mid \exists element' \in CA, g'(element) = element'\} .$$

Le résultat de cette opération peut être : (i) identification complète de l'interface - branche principale de la figure 11.10, (ii) identification partielle - branche a) de la figure 11.10, (iii) pas d'identification possible - branche b) de la figure 11.10.

$$InterfaceIdentification(NCA, CA) \rightarrow \{Full, Partial, Empty\}$$

Sur et sous ensemble d'interface

L'étape d'identification des interfaces *(Interface Identification)* peut aboutir à une identification partielle, qui peut se traduire par une différence de vocable et est traitée par une identification de synonymes. Si ce n'est pas le cas, la nouvelle interface peut être un sur ou un sous ensemble d'une interface existante, l'identification est réalisée par la fonction *Interface Is Subset ?*.

$$InterfaceIsSubset(NCA, CA) \rightarrow \{True, False\}$$

Recherche de similarité nominative et structurelle

Le contrôle *Naming issue* vérifie si le fragment de modèle NCA a la même structure qu'un ou qu'une combinaison[3] de *core assets* qui vérifient les contraintes d'interface.

$$NammingIssue(NCA, CA) \rightarrow \{True, False\}$$

Toujours en considérant l'exemple des diagrammes de classes, une comparaison de similarité entre deux structures est positive si les structures ont un même nombre de classes et les mêmes relations entre chaque classe (c.-à-d. composition, généralisation). La similarité relationnelle concerne les liens entre éléments source/cible similaires.

Dans la figure 11.11, le fragment de modèle (partie droite - R) partage la même interface que la composition de deux *core assets* existants (partie gauche - L). Leurs structures sont les mêmes et peuvent être déterminées comme analogues (similarité dans la structure).

L'algorithme est limité dans l'identification de structures similaires et des améliorations sont envisagées sur ce point (par ex. deux classes avec des attributs distincts, connectées avec une relation de composition pourraient être identifiées comme similaires à une seule classe contenant l'ensemble des attributs, de plus des équivalences entre motifs de conception pourraient également être intégrées. Par conséquence, le fait d'avoir des itérations entre les modifications dans le produit et l'identification des nouveaux *core assets* avec un impact limité est préféré (cf. figure 11.6 et le cas 1, avec une faible distance entre la base existante et les nouveaux fragments de modèles).

[3] fusionné par l'algorithme basé sur les signatures.

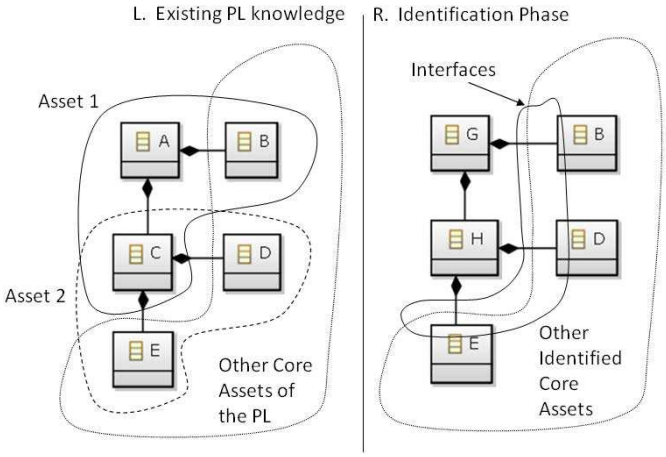


FIGURE 11.11: Exemple de structures similaires

L'opération variante *Interface Namming Issue* permet quant à elle de réaliser ce contrôle sur les artefacts d'interface uniquement.

$$InterfaceNammingIssue(NCA, CA) \rightarrow \{True, False\}$$

Détermination d'un opérateur logique de rattachement du nouveau *Core Assets*

À cette étape, une identification d'interface similaire est réalisée et complète, c.-à-d. qu'un ou plusieurs *Core Assets* existants possèdent une même interface, (identification de I_{NCA}). L'opération liée à *Variability operator* (Var_{op}) est appliquée aux *Core Assets* (liés au modèle de variabilité) correspondants au fragment de modèle.

$$Var_{op}(I_{NCA}) \rightarrow \{And, Or, Xor\} \ .$$

Dans le cas d'un *core asset* correspondant ou de plusieurs *core assets* liés à un même élément de variabilité, la fonction renvoie l'opérateur associé (de {And, Or, Xor}). Dans le cas de plusieurs éléments liés, une sélection ordonnée est offerte à l'utilisateur (sens opposé de la hiérarchie du modèle de variabilité).

Détermination de la présence d'un élément de la sélection sous un opérateur Xor

Dans le cas d'un opérateur Xor, une évolution par extension est uniquement possible si aucun variant n'est déjà sélectionné dans la configuration courante. L'opération *Element of the Xor* vérifie s'il existe d'autres éléments d'un point de variation ayant une logique *Xor* dans la configuration courante (et renvoie vrai ou faux).

$$OtherXorElement(Xor) \rightarrow \{True, False\}$$

Choix de gestion d'optionnalité

L'opération *In relation with a single asset* laisse le choix d'avoir un *asset* optionnel ou de transformer l'*asset* existant et le nouveau en une union des deux sous un opérateur *Or*. Comme l'illustre la figure 11.12, la sémantique d'incorporation d'un nouvel élément peut, selon l'utilisateur, créer un chevauchement de contenu dans les *assets*.

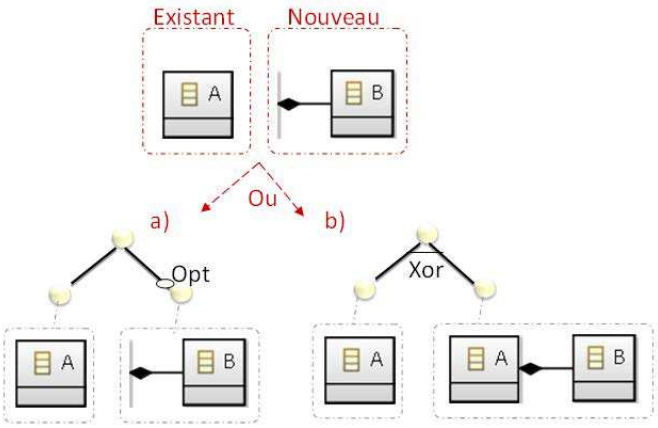


Figure 11.12: Exemple de choix d'expression de la variabilité

11.6 Le langage PLiMoS et l'évolution

Cette section présente une application du langage PLiMoS dans la gestion et la modélisation de l'évolution de l'espace de modélisation des *lignes de produits*.

11.6.1 PLiMoS et la traçabilité

Tel que présentée dans la partie précédente, l'expressivité relationnelle de PLiMoS couvre la modélisation de l'espace de modélisation délimité par le processus dirigé modèles de développement de la *ligne de produits*.

Le contenu sémantique de la modélisation avec PLiMoS est plus riche qu'une traçabilité "classique" mais peut s'y substituer au besoin. Ainsi lors des étapes d'identification de motifs de similitude, une analyse d'impact des modifications peut être menée et une propagation vers les différents niveaux d'abstraction réalisée.

L'étendue, ou importance de l'impact du changement se doit d'être évaluée. Le changement peut avoir un impact non local, une expertise de la propagation de ce changement est par conséquent à réaliser à un même niveau d'abstraction et entre niveaux distincts. Dans une approche classique, une estimation des coûts d'évolution est par la suite à envisager, de même qu'une évaluation des risques potentiels liés à cette évolution. De plus, la causalité induite par les relations intentionnelles aide à l'analyse d'impact et donc à la gestion de l'évolution.

L'étymologie de l'évolution prend son origine dans le latin *evolutio*, et approche des notions de "dérouler", "faire sortir quelque chose d'une autre chose", ou encore "expliquer". Dès lors quoi de plus commun que d'exprimer les déroulements temporels décrivant l'évolution des modèles de la LdPs par une continuité relationnelle. Un lien direct peut être définit entre PLiMoS et une gestion de configuration sémantique des modèles. Les relations concernant l'évolution décrivent un séquencement de moments de la *ligne de produits* et sont de type relation au cours du temps (*over time*, cf. chapitre 8). Ces points ne font pas l'objet de l'étude de la thèse, et malgré un intérêt certain, ne sont pas développés plus amplement.

11.6.2 Éléments relationnels dans le processus d'évolution par extension

Le processus d'évolution décrit dans la section précédente met en application des relations d'équivalence lors de la détection de similitudes. Dans la pratique et dans l'implémentation de l'algo-

rithme réalisé, toutes les relations ne sont pas systématiquement identifiées en tant que telles.

Par ailleurs, certaines relations caractéristiques du processus d'acceptation de l'évolution par extension sont déployées. Les dépendances émergentes sont créées par l'utilisateur durant la configuration et la dérivation. Les dépendances définies suite à l'analyse suivent les états (et donc la sémantique) exprimés ci-après :

- *emerging* : elles expriment des intérêts *ad-hoc* d'une nouvelle architecture de produit ;
- *inferred* : elles représentent une dépendance de configuration nouvellement détectée ;
- *confirmed* : les dépendances sont validées par les parties prenantes concernées ;
- *formalized* : elles sont formalisées dès lors qu'elles sont traduites dans les modèles de variabilité et les modèles relationnels, par l'action automatique de mise à jour.

La figure 11.13 présente un séquencement classique des quatre relations (émergente, inférée, confirmée et formalisée) dans le processus d'évolution par extension concernant des relations d'un modèle de *feature*. Le symbole "v" représente une validation de l'utilisateur, et un trait en pointillé une relation potentielle, enfin les traits en tirets représentent des associations interespace entre le modèle de *feature* et des *core assets*.

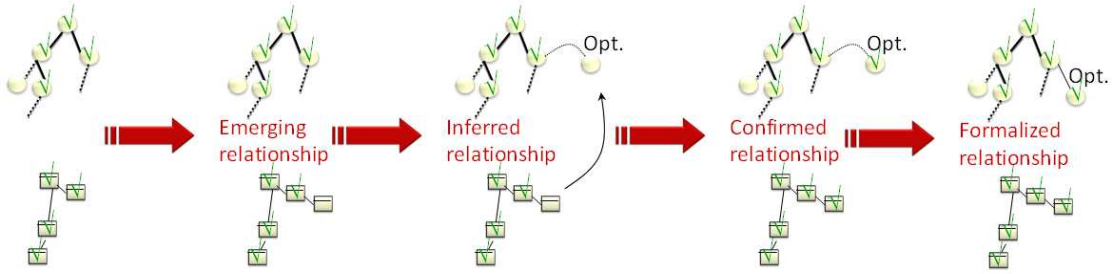


FIGURE 11.13: Succession de relations lors de l'évolution par extension

11.7 En résumé

Afin de pallier au manque de flexibilité et d'automatisme de l'évolution, la thèse propose un processus d'évolution incrémentale de la *ligne de produits*. L'objectif est de fournir un processus outillé pour faciliter la ré-injection des développements dans la *ligne de produits* et de réaliser une exploitation systématique des efforts de conception. La proposition marque un rapprochement entre le processus de dérivation de l'ingénierie de l'application et la gestion de la variabilité propre à l'ingénierie du domaine.

L'évolution ciblée est caractérisée comme une évolution par extension qui repose sur un processus semi-automatisé et outillé, centré sur la phase de dérivation des produits. L'étape de découverte des modifications, de la révélation et de l'incorporation de la variabilité repose sur un algorithme modélisé qui se base principalement sur un dictionnaire de vocabulaire métier, de l'homologie structurelle, et sur l'étude de la variabilité existante.

Quatrième partie

Application et applicabilité de la modélisation

"Es ist nicht genug, zu wissen, man muß auch anwenden ; es ist nicht genug, zu wollen, man muß auch tun." [a]

[a] Il ne suffit pas de savoir, il faut aussi appliquer ; il ne suffit pas de vouloir, il faut aussi agir.
— Johann Wolfgang von Goethe, *Wilhelm Meisters Wanderjahrem, 1796*

CHAPITRE 12

Modélisation de l'espace de modélisation, cas d'étude Thales

"L'information, c'est la néguentropie potentielle."
— C. de Beauregard

La section présente un cas d'étude Thales édulcoré en terme de modèle mais reprenant la structure, tant en terme de processus IDM, que de séparation des préoccupations. Les systèmes traités dans le cadre applicatif industriel sont des systèmes à logiciel prépondérant, et la description se fait suivant une perspective d'ingénierie logicielle (introduite dans la section 6.1.2, et figure 6.1, page 86). Le scenario décrit une mise en œuvre de *ligne de produits* de systèmes réactifs, à vocation d'évolution réactive. Le cas d'étude a été introduit très partiellement dans les publications [CMCJ11, CCMJ12] et fait l'objet d'un rapport technique en interne. Le cas d'étude entre dans le cadre de modélisation ARCADIA de Thales (introduit en section 2.2.1) et l'espace technologique de Thales Air Systems.

La section décrit tout d'abord la modélisation intentionnelle de l'espace de modélisation, puis les relations mises en place, pour enfin se focaliser sur la plate-forme conceptuelle architecturale. Certains détails non rapporté dans ce chapitre sont donnés dans l'annexe C.

12.1 Modélisation intentionnelle de l'espace de modélisation

Dans la suite, la description de l'espace de modélisation se concentre sur l'ingénierie du logiciel. L'hypothèse de départ de cette modélisation exprimant l'intention de modélisation de la conception et de la variabilité correspondante est une présence antérieure des modèles de conception sur les modèles de variabilité (du type approche extractive). L'intention de réalisation d'un produit, quant à elle, est traduite dans la modélisation relationnelle de la variabilité. La figure 12.1 fournit un aperçu de l'espace de modélisation utilisé dans le cadre de la thèse. La capacité de variabilité avec une perspective d'utilisateur final est modélisée dans l'espace du problème par un modèle *EVM*. Dans l'espace de la solution, les modèles de variabilité architecturaux et techniques ("Logical IVMs - *LIVMs*" ou "Physical IVMs - *PIVMs*", modèles internes correspondants à des niveaux d'abstraction *Logique* et *Physique*) sont multiples (séparation des préoccupations)

et se retrouvent en regard de *core assets* de conception (*LCA* ou *PCA*, suivant une séparation sur la nature logique ou physique des éléments) à différents niveaux d'abstraction. Les modèles de variabilité du niveau physique (*PIVMs*) sont des ramifications de nœuds d'éléments variables logiques et sont relatifs à des variations physiques (c.à-d. de la plate-forme).

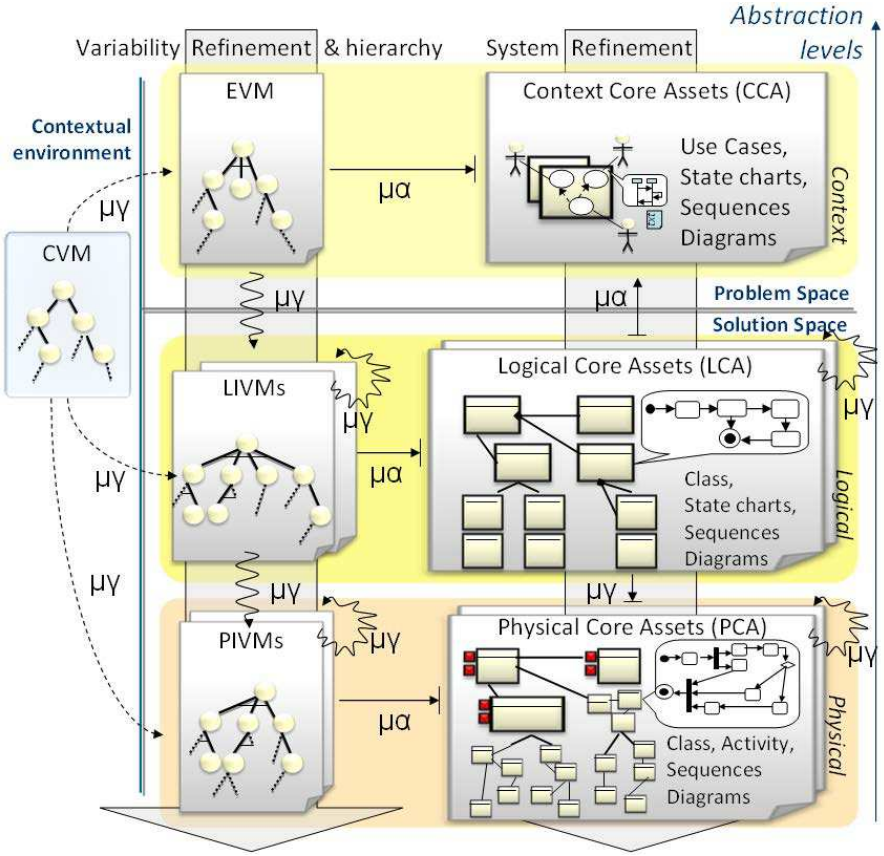


Figure 12.1: Organisation de l'espace de modélisation

12.1.1 Entre les deux espaces (problème et solution)

Les VMs réalisent une abstraction de la variabilité de l'espace des *core assets* à chaque niveau d'abstraction, les relations sont identiques à $\mu\#1$ (pour rappel, cf. page 101), et sont exprimées dans $\mu\#15$.

$$EVM \underset{\mu\alpha}{\longrightarrow\!\!|} CCA\,; \quad LIVMs \underset{\mu\alpha}{\longrightarrow\!\!|} LCA\,; \quad PIVMs \underset{\mu\alpha}{\longrightarrow\!\!|} PCA \qquad \mu\#15$$

Dans l'équation $\mu\#15$, comme l'*EVM* abstrait la variabilité de *CCA*, l'intention de l'*EVM* couvre partiellement celle des *CCA*. Il en va de même pour les relations entre les *LIVMs* et les *LCA*, et pour les *PIVMs* et les *PCA*.

12.1.2 Dans l'espace de la variabilité.

Les trois espaces introduits dans la section. 6.1.1 correspondent à une séparation des préoccupations de haut niveau. Concernant la variabilité et sa gestion, les principales parties-prenantes impliquées sont: les managers des ventes, les managers de la LdP, les ingénieurs systèmes et ceux du logiciel.

Entre le CVM et les deux autres types de VM

Tel que présenté par la relation $\mu\#8$ (cf. page 103) les variants externes sont induits par ceux du contexte environnemental. Dans la relation $\mu\#16$, bien que l'impact du *CVM* sur l'*EVM* soit un point clé lors du processus de dérivation, les modèles sont conçus avec des intentions totalement indépendantes en tête. Des relations identiques peuvent être définies entre (i) le *CVM* et les *LIVMs* et (ii) le *CVM* et les *PIVMs*.

$$CVM \xrightarrow[\mu]{} EVM\,; \quad CVM \xrightarrow[\mu]{} LIVMs\,; \quad CVM \xrightarrow[\mu]{} PIVMs \quad \mu\#16$$

Entre l'EVM et les IVMs

L'*EVM* est un modèle de variabilité externe du niveau d'abstraction *Contexte*, les relations $\mu\#3$ (cf. page 102) et $\mu\#4$ (cf. page 102) se translatent dans le référentiel de l'espace de modélisation pour pouvoir exprimer les relations $\mu\#17$.

$$LIVMs \underset{\mu\gamma}{\rightsquigarrow} EVM\,; \quad PIVMs \underset{\mu\gamma}{\rightsquigarrow} EVM \quad \mu\#17$$

Un point sur la modularité des IVMs

La relation $\mu\#18$ concerne la variabilité logique et physique (similairement à $\mu\#5$, décrite p. 102) : les modèles en vis-à-vis (c.-à-d. *LCA* et *PCA*) sont intentionnellement liés (raffinement de conception), et les *LIVMs* et *PIVMs* le sont également (la variabilité est complétée par le raffinement).

$$PIVMs \underset{\mu\gamma}{\rightsquigarrow} LIVMs \quad \mu\#18$$

Pour les VMs de type interne, la modularité œuvre à réduire la complexité accidentelle visible et à renforcer une séparation des préoccupations. L'approche proposée raffine la décomposition hiérarchique inhérente aux VMs avec une décomposition en préoccupations horizontales et verticales : décomposition en niveaux d'abstraction (*Logique*, *Physique* comme introduit précédemment), et pour un niveau donné, en préoccupations organisationnelles.

Considérons la décomposition en n IVMs pour le type *LIVM* (respectivement pour les *PIVMs*). Chaque $LIVM\,i$ (respectivement $PIVM\,i$), $i \in \{1, \ldots, n\}$, se concentre sur une préoccupation d'une partie-prenante qui peut potentiellement recouvrir une préoccupation d'un autre $LIVM\,j$, $j \in \{1, \ldots, n\} \wedge i \neq j$ (respectivement $PIVM\,i$). Conséquemment, certains éléments variables peuvent être équivalents et faire référence aux mêmes concepts et artefacts. Les relations $\mu\#19$ sont dès lors définies pour décrire ce phénomène.

$$LIVMi \underset{\mu\gamma}{\rightsquigarrow} LIVMj\,;\; PIVMi \underset{\mu\gamma}{\rightsquigarrow} PIVMj \quad \mu\#19$$

La prise en compte de la variabilité des espaces du problème et de la solution dans leur globalité, c.-à-d. l'*EVM* et l'union des *IVMs* ($\bigcup$ *IVMs*), résulte dans $\mu\#20$ (similaire à $\mu\#9$, p. 104). Chaque variant de l'EVM doit être exaucé par la variabilité de la solution.

$$EVM \underset{\mu\gamma}{\longrightarrow\!\!|} \bigcup IVMs \quad \mu\#20$$

12.1.3 Dans l'espace des *Core Assets*

Les relations $\mu\#6$, $\mu\#7$ (cf. page 102) décrivent les relations entre *CCA*, *LCA* et *PCA*.

L'approche telle que décrite met en œuvre de la variabilité positive. La variabilité positive se base sur la composition d'artefacts, qui sont par conséquent séparés dans l'espace de modélisation. À l'inverse d'une modélisation négative de la variabilité, la notion d'ensemble de *core assets* (*CASet*) est implicite. En considérant un produit p, ce dernier est défini dans le cas de la variabilité positive par la relation (a) de l'équation ci-après, et par la relation (b) pour la variabilité négative ($p \subseteq CASet$), S symbolisant les variants sélectionnés se conformant aux règles du modèle de variabilité.

$$(a) \quad p \triangleq \bigcup_{i \in S} CA_i \quad ; \quad (b) \quad p \triangleq \bigcap_{i \in S} CA_i$$

Considérons n *LCA* (respectivement *PCA*) ; un *LCAi* (respectivement *PCAi*), $i \in \{1, \ldots, n\}$, peut être combiné avec un autre *LCAj*, $j \in \{1, \ldots, n\} \wedge i \neq j$ (respectivement *PCAi*). Les relations $\mu\#21$ sont finalement obtenues.

$$LCAi \underset{\mu\gamma}{\rightsquigarrow} LCAj\,; \quad PCAi \underset{\mu\gamma}{\rightsquigarrow} PCAj \quad \mu\#21$$

12.1.4 Problématique de gestion de modèles.

De l'analyse de l'espace de modélisation d'une *ligne de produits dirigée par les modèles* résulte une élicitation de relations intentionnelles d'inter-modélisation, ces dernières révélant :

- *Des modèles interagissant :* dans le processus de l'ingénierie de l'application, des modèles infèrent sur la sélection de points de variation lors de l'étape de configuration du produit (relations : $\mu\#16$, $\mu\#17$, $\mu\#18$, et $\mu\#19$) et lors de la dérivation du produit (les mêmes que précédemment et la relation $\mu\#15$) ;
- *Des modèles se recouvrant :* dans l'espace de variabilité, les IVMs peuvent être composés pour un affichage et une représentation plus globale (relations: $\mu\#18$ et $\mu\#19$) ; dans l'espace des *core assets*, les *core assets* sont composés lors de la phase de dérivation (relation $\mu\#21$).

À noter que l'espace de modélisation présenté et les techniques employées ne définissent pas une utilisation de préoccupations transverses et de tissage d'aspects de modèle. Ce volet concerne le dernier axe de la variabilité multidimensionnelle, celui des préoccupations métier.

12.2 Modélisation avec les *Relationships* à une granularité plus fine

En s'intéressant à la modélisation de l'infrastructure de l'espace de modélisation avec une granularité fine, le langage PLiMoS vise à faciliter la gestion des relations entre artefacts durant la phase d'ingénierie du Domaine.

Les relations sont traitées telles des artefacts de première importance et élicitées dans le processus de *ligne de produits* ; certaines relations sont explicites et réifiées, d'autres sont inférées. Les relations de bas niveau sont définies comme des associations sémantiques entre artefacts de modélisation (entités, modèles totalitaires ou fragmentaires).

La figure 12.2 présente une représentation cartographique de la teneur des *relationships* mises en œuvre. Une relation de collaboration revêt plusieurs caractéristiques, et il est important de répondre aux questions telles que : quels sont les acteurs participant à cette collaboration ? Quelle est l'importance de cette collaboration pour les acteurs qui y participent ? Ces questions trouvent réponse dans la modélisation intentionnelle présentée en section précédente. Les questions "comment s'organise cette collaboration ?" et, "qu'est-ce qui est manipulé, échangé, partagé par cette collaboration ?" font partie des réponses à apporter par la structure de *relationships*. Le graphique renseigne sur (i) l'étendue de la collaboration (inter-LdP, inter-modèles, intra-modèle), (ii) la nature de celle-ci (mise à disposition - implication, échange - co-implication, ou partage - structure complexe), (iii) sa complexité en terme d'action à réaliser (induction, modification), ainsi que (iv) son moment d'action (conception, exécution, évolution).

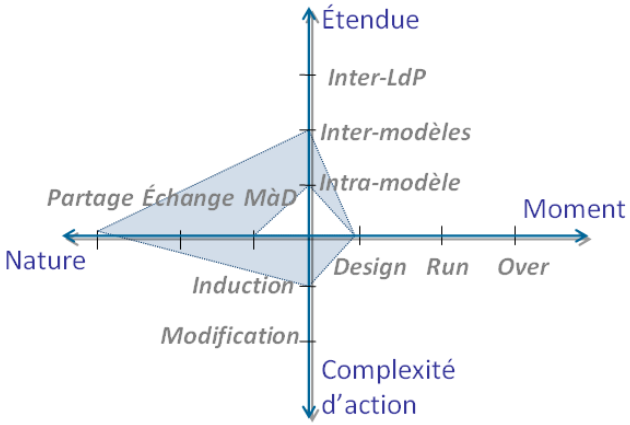


FIGURE 12.2: Contour des *relationships* nécessaires

La section fournie une description de l'infrastructure côté spécialisation, pour des détails sur le modèle de spécification à l'origine des modèles, voir l'annexe C, section C.2.

12.2.1 L'infrastructure établie sur la base des relations concrètes

L'intention d'obtention d'un produit s'accompagne, dans le cadre d'une *ligne de produits* de son objectif de dérivation d'un produit, se traduisant par une mise en relation opérationnelle des éléments de modèles. Les dépendances et relations entre artefacts de modélisation sont explicitées et réifiées dans un métamodèle pour composer l'ensemble des relations grain-fin de la modélisation de l'espace de modélisation par le langage PLiMoS. Modéliser des relations statiques aide à la connexion de nombreuses activités et outils comme des systèmes de gestion de version, de gestion de cohérence, etc. (cf. chapitre 9), un pré-requis à la maintenance et l'évolution de la *ligne de produits*.

Le sous-ensemble PLiMoS$_{specialisation}$ réalise un chevauchement avec le langage de variabilité et ceux des *core assets* (cf. fig. 6.8). De plus, l'approche utilise des modèles de *features* comme modèles de variabilité. Le langage PLiMoS$_{specialisation}$ spécifique à l'espace de modélisation du cas Thales est considéré comme étant constitué d'un ensemble de *relationships* intra-modèle de *features* (FMR), d'un ensemble de *relationships* inter-modèles (IMR) et d'un ensemble de *relationships* dérivées (DR) (présenté dans la section suivante).

$$PLiMoS_{specialisationThales} \triangleq \langle FMR, IMR, DR \rangle$$

Relations dans les modèles de variabilité

L'approche utilise des modèles de *features* comme modèles de variabilité, typés en modèles de contexte, externe, et interne. La sémantique des modèles de *features* utilisés figure dans la section introductive de ce document (section 3.3, page 44). Dans la suite de la présentation, les ensembles de *features* se distinguent par l'apposition en indice du type de modèle concerné, c.-à-d. {CVM,EVM,IVM}, par ex. F_{CVM} pour un ensemble de *features* de CVM ($F_{CVM} \cap F_{EVM} = \emptyset, F_{CVM} \cap F_{IVM} = \emptyset, F_{EVM} \cap F_{IVM} = \emptyset$).

Un modèle de relation de FM (FMR) est défini comme un 4-uplet $\langle E, IDC, CIDC, EDC \rangle$:

- *Consist-of* (χ): sont des relations de co-implication, binaires (1-to-1), homogènes, non-dérivées. Ces relations basiques de hiérarchisation sont raffinées par une intention donnée, soit de décomposition *part-of*, soit de généralisation *generalization*, et $E \subseteq F \times$ {generalization, decomposition}$\times F$;
- *Implication* (ι): ces relations d'expression de nécessité sont des contraintes de configuration non-symétriques mais transitives, $IDC \subseteq F \times F$, $IDC \subseteq CE$;
- *CoImplication* (I) : ce sont des contraintes symétriques d'implication, $CIDC \subseteq F \times F$, $CIDC \subseteq CE$; elles sont binaires *(1-to-1)*, homogènes, concrètes, comme les précédentes ; Cette relation peut être nécessaire dans les modèles techniques exprimant la variabilité de la solution, où une première décomposition hiérarchique n'est pas suffisante. Par exemple, les modèles de *features* avec une utilisation de *features* composites et abstraits utiles à une décomposition proche de la structure de la solution (pour des convenances de lecture et compréhension) peuvent impliquer un besoin de contraintes transverses bidirectionnelles ;
- *Exclusion* (η): décrivent des relations symétriques mais non transitives de mutuelle exclusion, $EDC \subseteq F \times F$, $EDC \subseteq CE$; ces relations sont de type *1-to-1*, homogènes, non dérivées ;

Manifestement, $E \cap IDC \cap CIDC \cap EDC = \emptyset$. Les relations de dépendances du modèle de *features* (ι, I, η), peuvent dans certains processus être associées à d'autres interprétations sémantiques pour un besoin précis, par ex. en fonction des usages, comme base pour des activités de dérivation ou de vérification de cohérence. Notamment, il est possible de distinguer des relations de dépendance liées aux opérateurs *And*, *Or*, *Xor* et *Optional* et aux variants.

Relations entre les modèles

Le sous ensemble PLiMoS$_{specialisation}$ est également concerné par l'expression des *relationships* inter-modèles de l'espace de modélisation. Un modèle *irm* de *relationships* inter-modèles (IMR) est un 3-uplet $\langle SR, IR, RR \rangle$ tel que :

- $EIR \subseteq \mathcal{P}(C_{CVM}) \times \mathcal{P}(F_{EVM} \cup F_{IVM})) \times \{implication, exclusion\}$ est un ensemble fini de relations d'*Environment-Induction* (σ). Les variants externes sont induits par ceux de l'environnement, et σ spécialise la relation intentionnelle μ#16. Afin d'illustrer cette relation, considérons trois éléments *features* du modèle de variabilité du contexte, F_{CVM1}, F_{CVM2}, et F_{CVM3}, et F_{EVM1}, F_{EVM2}, deux éléments *features* du modèle de variabilité externe.

$(\{F_{CVM1}\}, \{F_{EVM1}, F_{EVM2}\}, \iota) \in EIR$ représente le fait que l'élément F_{CVM1} implique F_{EVM1} et F_{EVM2}. De manière similaire, prenons l'exemple $(\{F_{CVM2}, F_{CVM3}\}, \{F_{EVM2}\}, \eta) \in EIR$ qui lui indique que la conjonction des éléments F_{CVM2} et F_{CVM3} exclut F_{EVM2} ;

- $SR \subseteq \mathcal{P}(C_{EVM}) \times \mathcal{P}(F_{IVM})$ est un ensemble fini de relations de *Satisfaction* (ζ). Les variants externes "marketing" sont satisfaits par des variants techniques internes des IVMs : les modèles internes satisfont les exigences et les besoins des modèles externes. La relation ζ est un raffinement de la relation intentionnelle $\mu\#17$;
- $ER \subseteq (F_{IVM\_L} \times F_{IVM\_L}) \cup (F_{IVM\_P} \times F_{IVM\_P})$ est un ensemble fini de relations *Identity* (Ξ), avec F_{IVM_L} le *feature* d'un modèle IVM à un niveau d'abstraction logique (F_{IVM_P} pour le niveau physique). La relation d'identité entre deux *features* signifie une similarité en respect de la variabilité et des concepts associés ; par conséquent, les *core assets* associés, si différents, doivent être totalement disjoints. La relation *Identity* est utilisée pour séparer des préoccupations diverses à un même niveau d'abstraction (horizontale). Ξ spécialise la relation $\mu\#19$;
- $SiR \subseteq V_{IVM\_L} \times V_{IVM\_P}$ encode l'ensemble des relations d'équivalence *Equivalence* (ξ). Un lien d'équivalence entre deux *features* signifie une similarité en regard de la variabilité mais pas des concepts. L'équivalence est utilisée pour séparer des préoccupations à différents niveaux d'abstraction (verticale) ; les *core assets* associés doivent être distincts et à différents niveaux d'abstraction. ξ spécialise l'intention $\mu\#18$;
- $RR \subseteq (C_{EVM} \times CA) \cup (C_{IVM} \times CA)$ est l'ensemble des relations de *Realization* (P). Un variant peut être réalisé par une entité de *core assets*(CA). Il n'y a pas de relations n-aires vers les *core assets*, cependant il faut garder en mémoire que le concept de CA représente un ensemble de *core assets*. P spécialise l'intention $\mu\#15$;
- $ReR \subseteq (CCA \times LCA) \cup (LCA \times PCA)$ est l'ensemble des relations de raffinement (*Refinement* - ρ). La relation décrit le raffinement technologique entre deux niveaux d'abstraction dans le processus modélisé de développement (elle spécialise les relations intentionnelles $\mu\#6$ et $\mu\#7$).

La figure 12.3 illustre le positionnement de certaines relations, et la figure 12.4 présente un extrait du métamodèle de spécialisation généré.

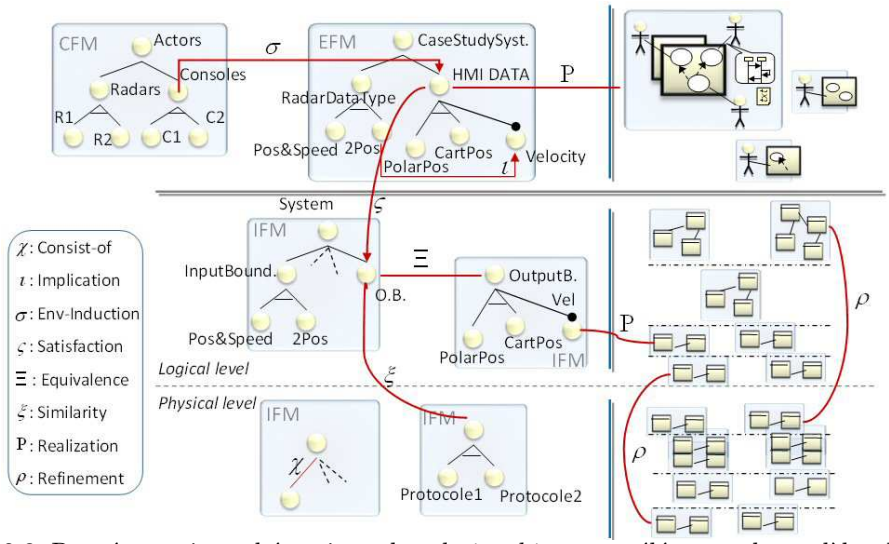


FIGURE 12.3: Représentation schématique de *relationships* entre éléments de modèles de l'espace de modélisation

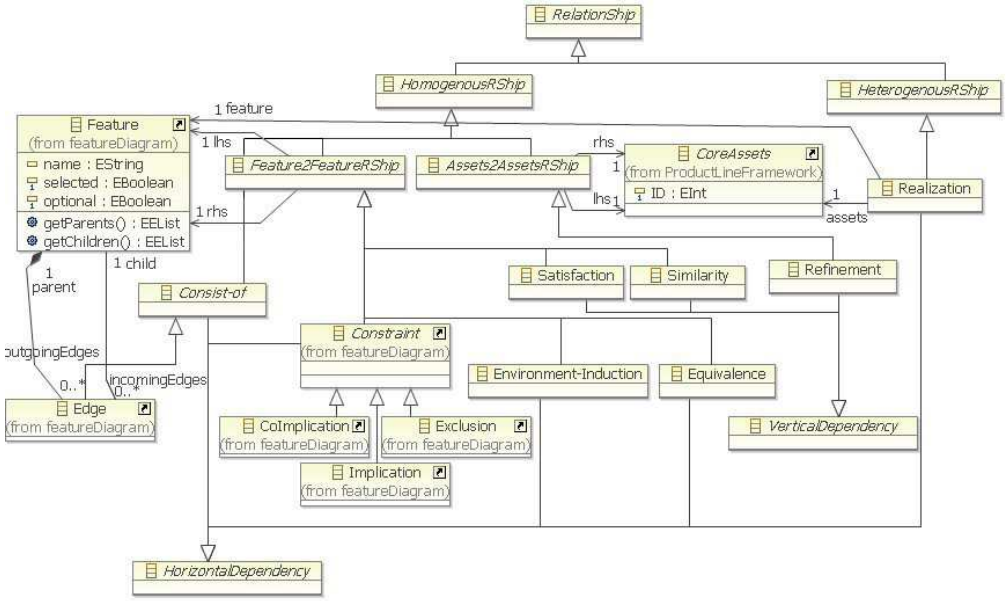


Figure 12.4: Extrait du métamodèle PLiMoS de spécialisation - *relationships* principales

12.2.2 L'infrastructure établie sur la base des relations dérivées

Cette section décrit un sous-ensemble de relations dérivées, et donc inférées tout au long du processus. L'intérêt d'expliciter des relations ne se limite pas aux relations structurelles comme présentées dans la section précédente, mais s'étend également aux relations dérivées, notamment pour des besoins de visualisation, vérification de cohérence, et dérivation de produits.

Des *relationships* nouvellement dérivées peuvent être inférées en composant des *relationships* structurelles, mettant en lumière un ensemble de connaissances jusque là implicites, et permettant de mettre en exergue des incohérences entre modèles non évidentes. L'inférence de ces relations aide à renforcer la cohérence sémantique entre les ensembles de modèles reliés. Ces relations sont réifiées dans un sous-paquetage du métamodèle et introduites comme dérivées, volatiles et *transient*.

Les *relationships* intra-modèle des modèles de *features* ont des impacts sur les autres modèles de l'espace de modélisation. La figure 12.5 (a) et (b) représente le fait que les relations χ avec des opérateurs entre C (un *feature* composé) et S_i (les sous-*features*, i identifiant les *features* distincts par un indice, $M \subseteq \{1, \ldots, n\}$) correspondent à des relations de co-implication ou d'exclusion. "And-χ" traduit une co-implication (I) entre les sous-*features*, "Xor-χ" relie les sous-*features* par des relations d'exclusion (η), et "Or-χ" n'induit aucune relation particulière de dépendance entre les sous-*features*.

Concernant l'EVM et les IVMs, des contraintes de conception peuvent apparaître dans la solution, qui ne sont *a priori* pas corrélées avec le modèle externe. Des relations "Design-ι", "Design-I", "Design-η", sont ainsi dérivées des contraintes des modèles de variabilité internes, et transposées dans la vue externe.

En étendant la description aux dépendances inter-modèles, (c) montre une transposition d'une relation *Implication* depuis le domaine des *core assets* en développement, définissant par conséquent une dépendance d'implication "Core-ι". De manière similaire sont spécifiées les relations

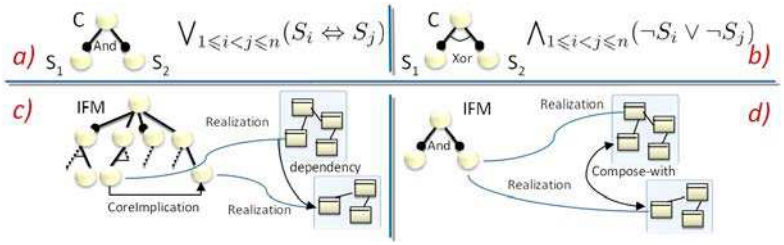


FIGURE 12.5: Des exemples de relations dérivées

"Core-I" et "Core-η". Une connaissance de l'algorithme de composition des *core assets* (dans le cas d'une variabilité positive) rend possible le déroulement d'analyse sur la base des *core assets* et permet l'inférence de contraintes dans les IVMs. Les relations "core" sont dérivées et montrées dans des vues spécifiques, et, si confirmées par l'utilisateur, peuvent être transformées en des relations intra-modèles de variabilité interne.

- $DCE \subseteq F_{EVM} \times DGCT \times F_{EVM}$ est l'ensemble des contraintes de conception, avec Design Graphical Constraint Type étant $\{\iota$,I, $\eta\}$, et $DCE \cap CE_{EVM} = \emptyset$;
- $CCE \subseteq F_{IVM} \times CGCT \times F_{IVM}$ est l'ensemble des relations de contraintes "core", avec CGCT - Core Graphical Constraint Type - qui est $\{\iota$,I,$\eta\}$, et $CCE \cap CE_{IVM} = \emptyset$;

D'autre part, la vérification de cohérence de modèles de *features* induit l'émergence de relations "Conflict". Cette relation est symétrique, non réflexive et non transitive.

L'élicitation systématique des relations dérivées, des absences de liens permettent de statuer sur la non satisfaction de certains *features*, ou de core assetss non employés (non connectés à aucun *feature*), et ceci grâce à des techniques externes de computation (par ex. solveurs SAT).

Un autre processus nécessite des relations dérivées, la dérivation de produits de l'ingénierie de l'application. La figure 12.5 (d) montre une relation symétrique "compose-with" qui est inférée à partir de relations *Implication* et *Resolution* ("ι"s et "P"s). En conséquence, la spécification de configuration d'un produit est définie par l'ensemble des relations *Implication* & *CoImplication*, et la spécification de dérivation de produits par un ensemble de relations "compose-with" (specialisant μ#21).

12.2.3 Une synthèse et classification des *relationships* utilisées

Une relation peut être caractérisée par le critère du niveau d'abstraction. Ce critère distingue les relations verticales (raffinement selon Anquetil *et al.* [AKM+10]) si les relations sont transverses aux niveaux, des relations horizontales (similarité pour Anquetil *et al.* [AKM+10]) si elles se limitent à un niveau donné. Un autre critère concerne la nature homogène ou hétérogène des relations. Homogène entre les modèles de variabilité, ou dans les *core assets*, ou hétérogène entre les deux types de modèles (correspondant à la dimension de variabilité de [AKM+10]).

Concernant les relations dérivées, dont le métamodèle n'est pas représenté ici, c.-à-d. les relations dérivées :

- les relations homogènes horizontales dérivées sont : "Design-ι", "Design-I", "Design-η", "Core-ι", "Core-I" and "Core-η" ;
- La relation "Compose-with" est homogène mais ne peut être classée avec le critère vertical/horizontal.

12.3 Considérations architecturales appliquées à l'espace de modélisation

La complexité additionnelle apportée par la gestion des variants peut alourdir la tâche des architectes logiciels qui doivent aboutir à une solution logique pour une famille donnée. Une gestion propre de la variabilité est également importante pour le succès de l'architecture, car la variabilité permet à l'architecture d'être instrumentée de manière à favoriser l'évolution. Il est donc indispensable d'organiser l'espace de modélisation de la *ligne de produits* autour d'une plate-forme conceptuelle basée sur le patron d'architecture.

Le cadre de développement des applications suit toujours la méthodologie présentée dans [Voi08]. Les activités de modélisation sous-tendent les développements système et logiciel pour les exigences, la conception conceptuelle et détaillée et l'implémentation. Les niveaux d'abstraction définis sont pour rappel : *Contexte*, *Logique* et *Physique*.

12.3.1 La plate-forme conceptuelle

Le patron d'architecture présenté précédemment introduit un ensemble de conventions et règles de par ses couches et partitions, permettant d'organiser l'architecture par des modules cohésifs. La figure 12.6 illustre la situation de la plate-forme conceptuelle dans son environnement.

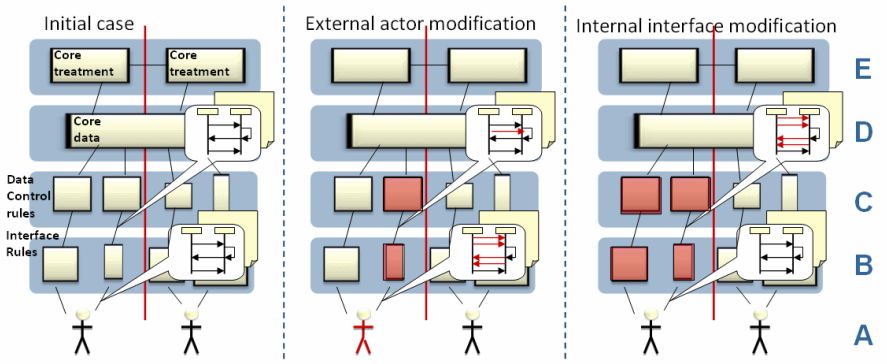


Figure 12.6: Plate-forme conceptuelle pour les Lignes de Produits.

Le patron organise la variabilité et sépare les préoccupations métier de celles d'interface. Conséquemment, les points de variation de la ligne de produit sont distingués par leur degré de stabilité. Les instables, induits par l'environnement, sont restreints, autant que faire se peut, dans les couches d'interface. À l'inverse, les couches métier (*Core expertise*) contiennent des points de variations et variants stables.

La figure 12.7 représente la variabilité au niveau *Logique* d'un cas d'exemple simplifié afin d'illustrer cette section (avec des stéréotypes "variation" et "variant", des contraintes d'inclusion et d'exclusion ; le profil manipulé est inspiré de [ZHJ03]). Sa taille permet de clarifier les couches ABCDE et le type de variabilité présent dans les *lignes de produits* de Thales. L'exemple ne permet pas par contre d'illustrer les partitions. Les systèmes de la famille de contrôle antiaérien reçoivent en entrée des données de capteurs (généralement des radars, la figure 12.7 en représente deux : Radar1 qui envoie vitesse et position, Radar2 qui envoie deux positions), effectuent des traitements (calcul de position et optionnellement de vitesse au besoin) et renvoient des données à l'utilisateur (via une console, la console1 affiche des données en coordonnées polaires, l'autre en

cartésiennes par exemple). La variabilité est limitée : au type de données en entrée (délivrées par des radars externes), au type de traitement nécessaire inférant, à la présence/absence de mémoire interne et enfin, au type de données produites et délivrées. Notons que la variabilité provient, en grande partie, des acteurs externes comme dans nos applications réelles.

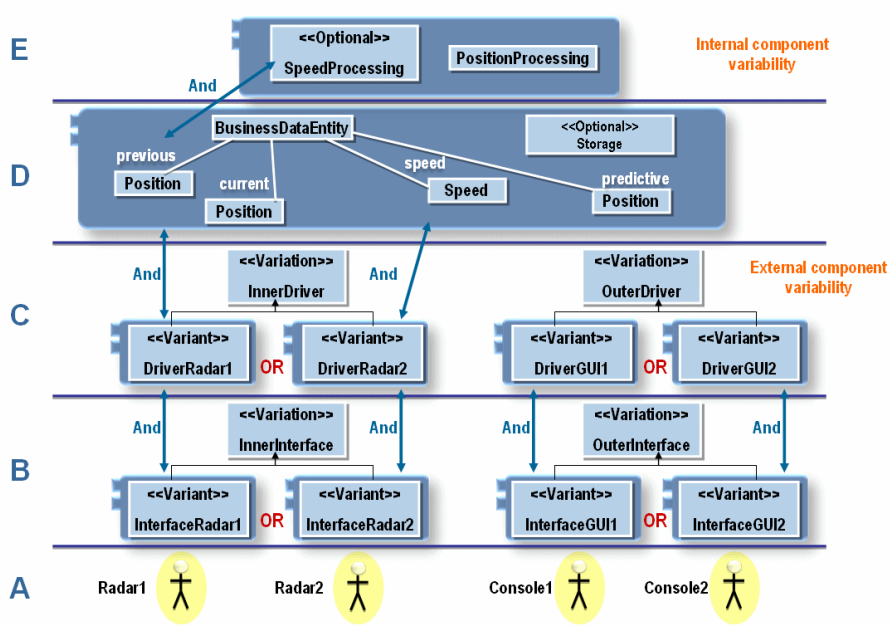


FIGURE 12.7: Une architecture conceptuelle réceptacle de variabilité.

12.3.2 Une structuration de l'espace de modélisation autour de la plate-forme

La plate-forme conceptuelle introduite dans la section précédente renforce la structure de la *ligne de produits*. Cette section décrit, de manière globale, l'organisation de la LdP autour de cette plate-forme. L'objectif n'est ni d'expliquer l'adoption de la LdP, ni de décrire son l'évolution ; la problématique de l'extraction de la variabilité dans des modèles de variabilité ne fait pas l'objet de ces travaux.

La figure 12.8 situe la plate-forme au niveau *Logique* (délimitée par un rectangle discontinu) et l'organisation de l'espace de modélisation autour de cette dernière. L'organisation de l'espace de modélisation reflète le processus IDM, avec le découpage en niveaux *Contexte*, *Logique*, et *Physique*), ainsi que la structure architecturale apportée par le patron (à partir du niveau logique).

Au niveau *Contexte*, les "Use-Cases" représentent des fonctionnalités significatives et centrales du système final. Au niveau *Logique*, les éléments de réutilisation de base ("Core assets") sont des classes ou des composants logiques et leurs relations. Enfin, au niveau *Physique*, les éléments de base sont des composants et leurs relations. Un développement orienté composant n'est pas, en lui-même, suffisant pour permettre d'obtenir un modèle produit valide et cohérent par dérivation. Les modules réutilisables ("Core Assets") sont caractérisés par leur niveau d'abstraction et leur affiliation à un niveau d'abstraction, et donc rangés et préservés suivant cette classification. Les modules de niveau métier peuvent garder une variabilité interne (la résolution et fixation du point

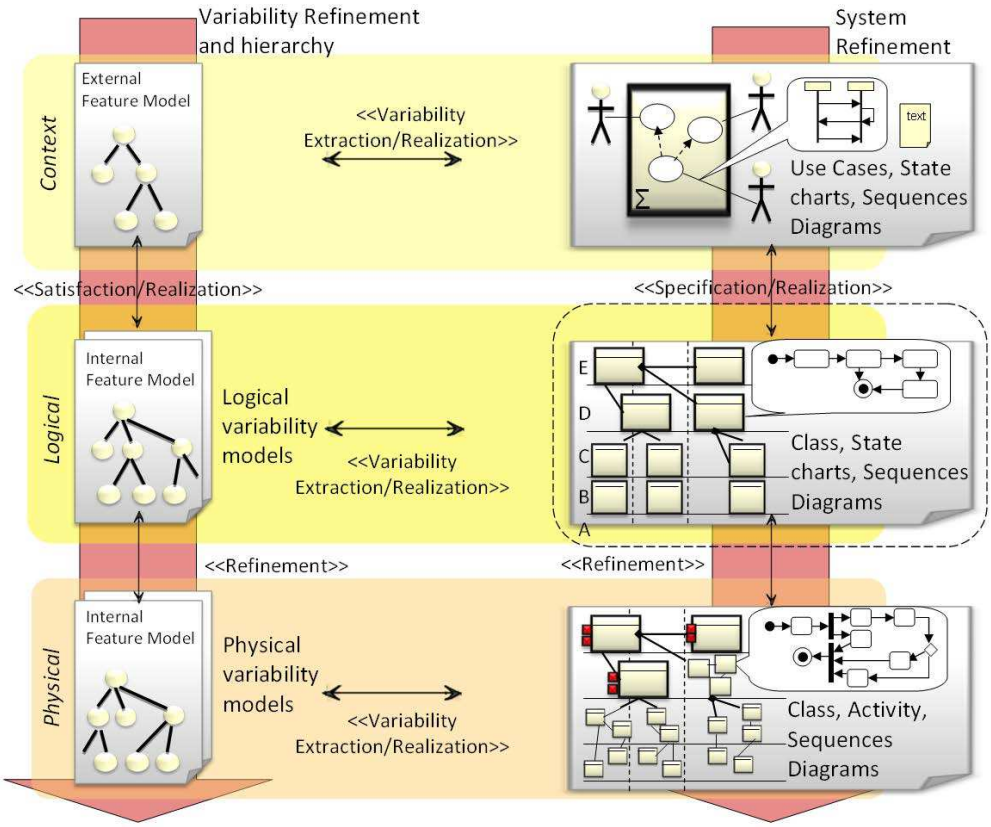


Figure 12.8: Vue d'ensemble : structuration de l'espace de modélisation.

de dérivation se faisant lors de la dérivation), exemple figure 12.7, alors que ceux d'interface ont une granularité plus fine.

La figure 12.7 ne représente pas les relations introduites dans la partie précédente, mais les symbolise grossièrement. Les artefacts de modélisation sont reliés entre eux par : (i) des relations d'extraction / réalisation de la variabilité (*"Variability Extraction / realization"*), (ii) des relations de satisfaction / réalisation (*"Satisfaction / Realization"*), (iii) des relations de spécialisation (*"Specification / Realization"*), et (iv) des relations de raffinement *"Refinement"*.

Du point de vue *lignes de produits*, les modèles de variabilité internes réalisent la spécification de la variabilité externe (liée au *Contexte* - espace du problème) au travers de liens vers des modules logiques. La translation du *Contexte* vers le *Logique* correspond à une résolution technologique de la variabilité.

La décomposition des modèles de *feature* et les dépendances de *satisfaction* permettent de connecter les objectifs commerciaux et la description de l'architecture et son implémentation. Une amélioration de la filiation des exigences vers les détails logiques du système en développement peuvent éviter l'apparition de déviations architecturales significatives.

12.3.3 L'impact du patron d'architecture dans l'espace de la variabilité

L'espace de modélisation est dans le cas présent découpé en modèles de variabilité externe et interne (sur deux niveaux d'abstraction), cf. section 12.1. Dans l'exemple qui suit, les modèles de

variabilité employés sont des modèles de *features* de type *External Feature Model - EFM* (précédemment noté de manière générique EVM) et *Internal Feature Model - IFM* (précédemment IVM).

Toujours sur le même cas d'exemple (introduit section 12.3.1), la figure 12.9 - a) représente le modèle de *feature* externe. Ce modèle de caractéristiques ne contient pas intrinsèquement dans sa hiérarchie de contraintes de conception propres à la solution architecturale.

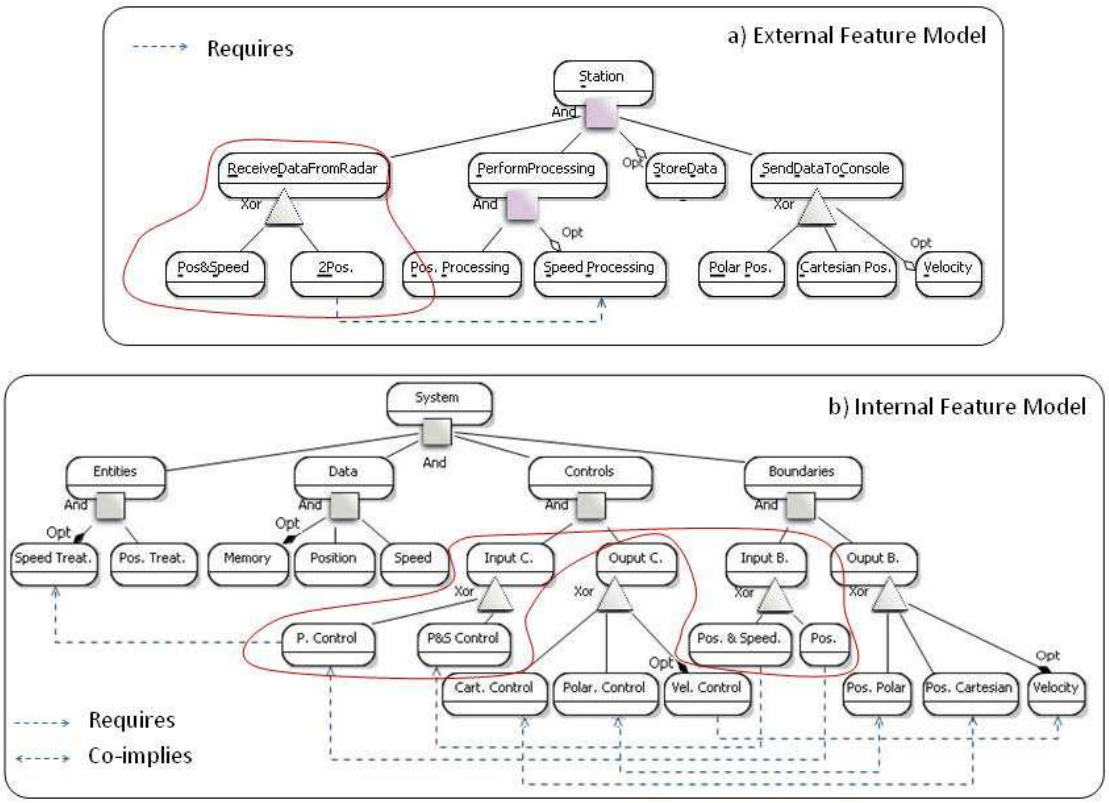


FIGURE 12.9: Modèle de *feature* du cas d'exemple

Le motif architectural est abstrait dans le modèle de variabilité de la solution. À partir du niveau *Logique* les modèles logiciels dépendent du patron spécifique présenté dans la section 10.3 (soit à partir des IFMs dans l'espace de la variabilité). La variabilité est, dès lors, organisée en diagrammes avec des préoccupations de découpe en couches ABCDE et partitions. Les IFMs contiennent autant de sous-ensembles délimités que de couches d'architecture. Le référentiel couche/partition côté architecture est adapté à la description en diagrammes de caractéristiques et les partitions sont exprimées telles des contraintes de co-implication entre *features*.

À noter, dans l'exemple de la figure 12.9 - b) la présence de *features* composites, permettant d'améliorer la compréhension de l'ingénierie. Les *features* composites ne sont pas forcément abstraits (les *features* abstraits permettent de structurer le FM sans avoir d'impact sur l'implémentation). Les IFMs, plus techniques, sont construits par et pour les ingénieurs, qui ont leurs préoccupations en tête. Dans ce type de modèle de variabilité, une décomposition architecturale apparaît, par ex. la figure 12.9 - b) décrit des *features Entities*, *Data*, *Controls* et *Boundaries* qui proviennent de l'utilisation de la plate-forme conceptuelle. Remarquons que la profondeur de l'arbre diffère suivant les quatre principales branches : le motif d'architecture organise la variabi-

lité et la concentre vers les couches *Controls* et *Boundaries*. Dans la réalité, il n'y a pas forcément un variant *Controls* correspondant à un variant *Boundaries*.

12.3.4 Discussion

La méthodologie et le patron d'architecture proposés, bien que résultant de besoins rencontrés dans le domaine de la défense, ne se limitent pas à ce dernier : le nombre de partitions et l'organisation pouvant être différents en fonction des préoccupations métier ciblées. De plus, dans l'absolu, il est possible d'ajuster les contraintes délivrées au niveau logique par le patron au niveau physique, c.-à-d. qu'une adhérence stricte au patron n'est pas obligatoire, elle doit cependant être motivée et documentée.

Une variation de l'approche plus importante est possible. La plate-forme conceptuelle établie par le patron est considérée comme une caractéristique de la *ligne de produits*. Par conséquent, une autre LdP, adressant des préoccupations pour un domaine distinct, mettrait en œuvre un autre patron. Par exemple le *"Pipes and Filters"* pourrait être une plate-forme conceptuelle pour une famille de systèmes orientés traitement de flux de données. La structure amenée par le motif d'architecture révèle la variabilité de l'architecture, dans le cas du patron ABCDE, au niveau des couches. Un patron d'architecture guide l'identification de la variabilité en regard de ces concepts manipulés.

Le modèle de variabilité interne (dans le cas de notre exemple l'IFM) fournit des renseignements, par delà sa structuration par un patron d'architecture, sur la variabilité des concepts du patron. Une grande profondeur d'arbre amène à s'interroger sur l'adéquation entre le domaine et le patron sélectionné.

Les bénéfices lors de la dérivation

Dans l'approche, l'architecture de la LdP et sa variabilité sont organisées en termes de *features*, offrant une canalisation pas à pas du processus de dérivation de produits.

La dualité variabilité du problème et de la solution est renforcée par la définition de la plate-forme conceptuelle d'architecture qui met en exergue les contraintes architecturales dans l'espace de la solution et dans l'espace de modélisation (modèles de variabilité et de *core assets*). La transition entre les *features* "marketing" (externes) et l'implémentation se retrouve favorisée.

Alors que dans les approches traditionnelles, les décisions de conception sous-jacentes se retrouvent habituellement enfouies dans les composants de réalisation, l'approche présentée ici, renforcée par l'utilisation d'une plate-forme conceptuelle basée sur l'explicitation d'un patron réduit la complexité cognitive du processus de dérivation, et renforce la validation du produit par construction.

De plus, les modèles de variabilité internes, proches de la solution technique, incorporant des contraintes architecturales, favorisent l'obtention d'un modèle valide lors de la dérivation.

En conclusion, le motif d'architecture est garant de la cohérence de la *ligne de produits* (de par son intégration à l'ingénierie du domaine), et permet une dérivation d'autant plus aisée que contrainte et organisée (lors de l'ingénierie de l'application).

La plate-forme conceptuelle et l'évolution de la LdP

La plate-forme conceptuelle se base sur un patron d'architecture qui borne le périmètre de la LdP, néanmoins il paraît légitime de s'interroger sur l'avenir de l'architecture lorsque l'évolution sort significativement des normes imposées par le patron.

Considérons AP comme étant l'ensemble des patrons d'architecture et A, ensemble des architectures. Considérons maintenant deux patrons $x, y \in AP$. Si une architecture $a \in A$ est

conforme à un patron x, cette dernière doit être modifiée de manière à ce qu'elle soit conforme aux patrons x et y. Il faudrait conséquemment définir un opérateur $\overline{X} : (A \times AP) \times AP \rightarrow A$ tel que :

$$(\overline{X}(a, x, y) = b \Rightarrow b \text{ conforms to } y)// \wedge (\overline{X}(a, x, y) \text{ defined such that } a \text{ is conform to } x);$$

Se pose dès lors la question de la compositionnalité de deux patrons (et donc de la possibilité de définition d'un tel opérateur) et la question de l'utilité de rester dans le périmètre de la LdP existante ; ces questions ne sont pas à l'ordre du jour de la thèse, néanmoins, il peut être intéressant de rappeler la possibilité plus pragmatique de définir une autre LdP, utilisant les mêmes *core assets* disponibles dans un réceptacle commun (*repository*), tel illustré par la figure 12.10.

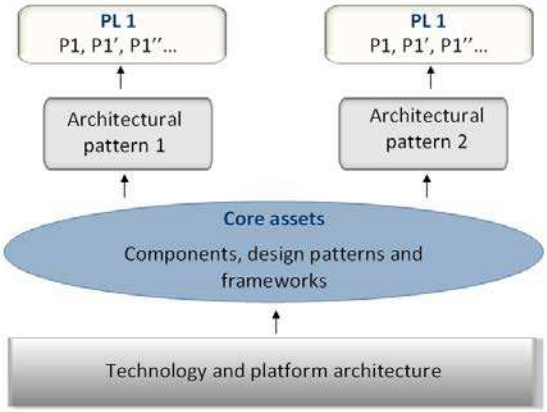


FIGURE 12.10: *Core assets*, patrons d'architecture et *lignes de produits*

La préoccupation horizontale de partitionnement logique permet de séparer des grandes fonctionnalités du système (fonctionnalité métier et interfaçage), alors que la séparation verticale fait référence à la structuration en couches ou niveaux permettant de séparer les éléments de la plateforme d'implémentation des niveaux hauts, typiquement des niveaux applicatifs représentés par des exigences métier.

Un découplage des composants et fonctionnalités permet à chaque artefact d'être remplacé ou amélioré isolément. La notion de composant ne doit pas se limiter à sa vision physique d'implémentation, mais englober une description logique conceptuelle, à l'exemple d'une découpe en acteurs dynamiques au niveau conceptuel, correspondant à la réalisation de services et fonctionnalités, avec une variabilité d'implémentation disponible pour adresser plusieurs plate-formes.

12.4 Conclusion

L'outillage réalisé permet une validation de l'approche par une étude de cas industriel dans l'ingénierie du logiciel et montre l'adéquation et la cohérence de la méthodologie proposée avec le processus de développement en place à Thales (ARCADIA) et avec la suite outillée présente à Thales Air Systems.

Sur un plan purement technique, le langage PLiMoS et la plate-forme conceptuelle ABCDE sont applicables, bien que certains développements restent à réaliser pour conserver certains outils propriétaires.

Par ailleurs, il est important de noter l'apport de la méthodologie, autant par la proposition de séparation des préoccupations générique et des relations sémantiques associées, que par la

proposition architecturale, qui répondent à des besoins clairement établis d'organisation et de gestion de la complexité. En pratique, la méthodologie a permis de lever les bonnes questions quant à l'organisation de la *ligne de produits* autour de l'architecture. L'approche apporte une aide à la compréhension et centre la réflexion sur les points clés à analyser.

Au delà du cas d'étude présenté dans la section, la clarification de l'information relationnelle a amené à des études sur l'organisation et le travail collaboratif entre différentes ingénieries (système et logiciel) afin de préciser les échanges de variabilité que les facilités de PLiMoS permettent de gérer (cohérence), et de permettre, à terme, une amélioration de la productivité.

CHAPITRE 13

Application, outillage et discussion

"Πάντων γὰρ ὅσα πλείω μέρη ἔχει καὶ μὴ ἔστιν οἷον σωρὸς τὸ πᾶν, ἀλλ΄ ἔστι τι τὸ ὅλον παρὰ τὰ μόρια." [a]

[a] il va une cause à l'unité de ce qui a plusieurs parties dont la réunion n'est point une sorte de monceau, de tout ce dont l'ensemble est quelque chose indépendamment des parties - trad. A. Pierron et C. Zevort

— Aristote, *Métaphysique, livre VIII*

Dans le cadre de l'approche, autour du langage PLiMoS et du processus d'évolution incrémentale, un certain nombre de prototypes outillés et de modèles de réalisation ont vu le jour, cette section en fait l'écho.

Concernant les approches de validation mises en œuvre dans les travaux de recherche orientées génie logiciel, il existe plusieurs types de validation : l'analyse, l'évaluation, l'expérience, et l'exemple. Suite au cas d'étude Thales présenté dans le chapitre précédent (qui est en réalité le dernier dans l'ordre chronologique de réalisation), ce chapitre introduit quatre cas d'études issues de la littérature. Par la suite un récapitulatif de l'outillage réalisé est proposé, pour enfin faire place à une conclusion sur la partie validation.

Pour de plus amples détails sur les propos introduits dans ce chapitre, le lecteur est invité à se référer à l'annexe C.

13.1 Modélisation de l'espace de modélisation, cas d'étude de la littérature

La section introduit quatre cas d'étude de la littérature, sélectionnés car présentant diverses caractéristiques distinctes, permettant de confronter le langage PLiMoS aux faits d'études externes. La figure 13.1 offre un aperçu graphique de l'étendue des collaborations et relations employées dans les quatre cas traités ci-après. Ces cas d'étude sont centrés sur l'espace de variabilité et, de ce fait, aucune description de l'architecture, ni proposition de plate-forme conceptuelle sur la base d'un patron n'est abordée.

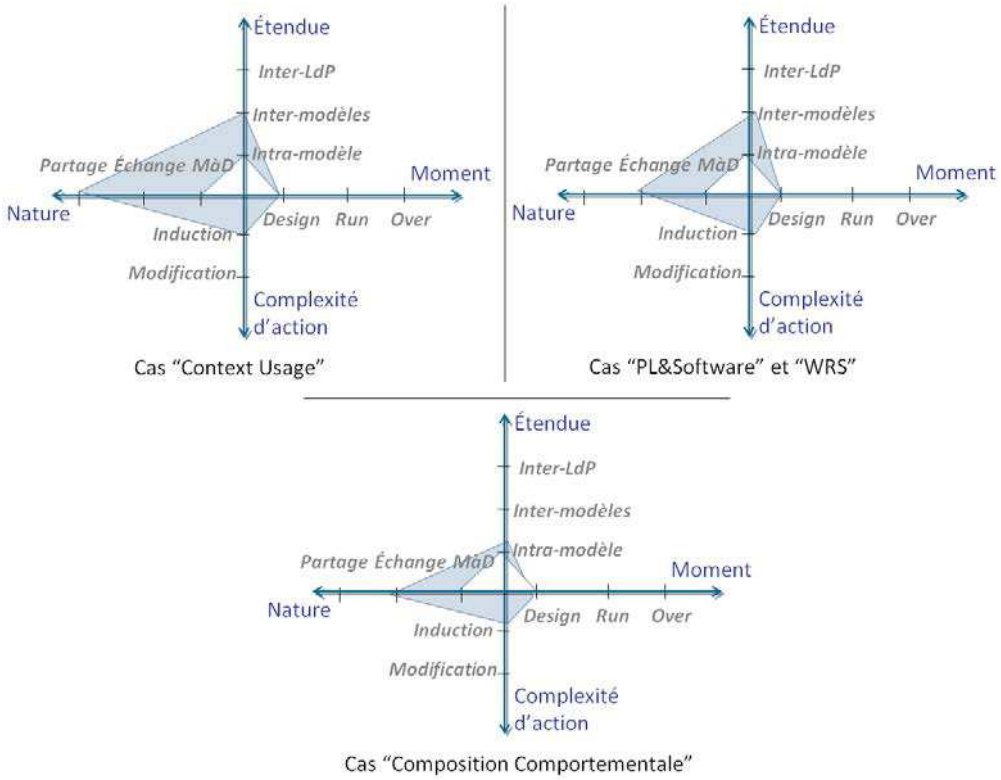


Figure 13.1: Cartographie des collaborations des quatre cas d'étude.

13.1.1 Cas d'étude "Context Usage"

Les travaux de Lee et Kang [LK10] mettent en avant le besoin d'avoir un modèle de *features* décrivant les contextes d'usages indépendamment du modèle de *features* principal (structurel de la variabilité de la LdP), guidant ce dernier lors de la dérivation de produits. L'exemple concerne une *ligne de produits* de logiciels de contrôle d'ascenseur (*elevator control software* - ECS).

Brièvement, le modèle de *features* employé met en œuvre plusieurs relations hiérarchiques (*aggregation, generalization, implemented-by*), ainsi que les opérateurs de variabilité *alternative* (Xor), Or, combinés aux dépendances de configuration *requires* et *excludes*. En outre, plusieurs catégories de modèles apparaissent dans l'approche, pour les citer :

- Le "*Usage Context Variability Model*" (UCVM) ;
- Le "*Quality Attribute Variability Model*" (QAVM) ;
- Le "*Product Feature Variability Model*" (PFVM) ;
- Le *mapping* "*UC-QA*" ; Le *mapping* "*QA-PF*" ; Le *mapping* "*UC-PF*" ;

Relevons un détail relationnel particulier avec le *mapping* "*QA-PF*", de type ternaire associant des *features* du QAVM, ceux du PFVM, et un ensemble de poids d'impact appartenant à make,help,hurtr,break. Ainsi, $QAPFR \subseteq F_{CPFVM} \cup F_{OEPFVM} \cup F_{DTPFVM} \cup F_{ITPFVM}) \times F_{QAVM} \times R$, l'ensemble des relations inter-modèles, inter-préoccupation, d'implication (implication de F_{QAVM} par F_i avec indication d'attribut de qualité), horizontales et concrètes, avec attributs de qualité associés ($R = \{make, help, hurt, break\}$), entre les modèles PFVMs et QAVM.

En supplément de la représentation diagrammatique de relation "filaire", une table ternaire

permet de renseigner la relation *QA-PF*, telle que représentée dans la figure 13.2, avec en ligne les éléments de $QAVM$, en colonne ceux des $PFVMs$, et à l'intersection un attribut qualifiant la relation.

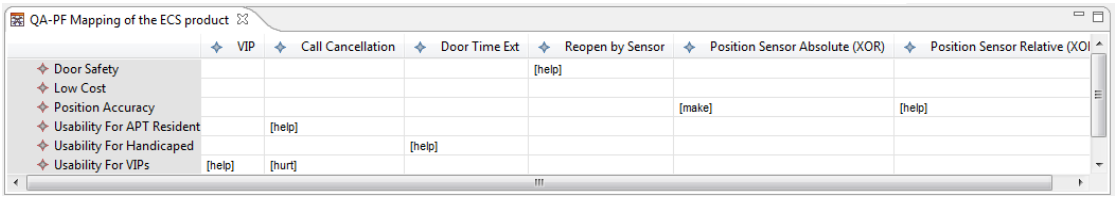


FIGURE 13.2: Extrait de syntaxe concrète - représentation d'une relation $QA - PF$.

Un autre exemple de relation, "*UC-PF*", elle de type ternaire et n-to-m, octroie la possibilité de décrire en un *mapping* entre l'élément déclencheur dans $\mathcal{P}(UC)$, ainsi que l'élément (ou les éléments) requis et l'élément exclu (ou les éléments) dans $\mathcal{P}(PF)$. La syntaxe graphique adoptée pour représenter des relations n-aire se rapproche de celle connue pour représenter des composants, laissant apparaître un ensemble d'intrants et d'extrants.

13.1.2 Cas d'étude "PL & Software variability

Metzger *et al.* [MPH+07] présentent une organisation de l'espace de variabilité caractérisée par une séparation de la variabilité de la Ligne de Produits (exprimée en langage OVM [PBL05]), de celle du logiciel (exposée en langage de *features*). Afin de décrire la manière dont la variabilité de la LdP est réalisée par le logiciel, les auteurs introduisent des relations nommées "*cross-model links*" (X-links) de sorte que lorsqu'un variant du modèle OVM est sélectionné, tous les *features* associés le soient également. Ce cas d'étude est traité avec le langage PLiMoS.

13.1.3 Cas d'étude "WRS"

L'espace de modélisation proposé par *et al.* [TTBC+09] étend le cas précédent et définit trois catégories de variabilité représentant le *World context* (W), les exigences (R), et les spécifications (S). Les auteurs proposent d'utiliser les relations "*cross-model links*" (X-links) entre les modèles de variabilités. Ce cas d'étude est traité avec le langage PLiMoS.

13.1.4 Cas d'étude composition comportementale

La composition comportementale dans le cadre de l'IDM et plus particulièrement appliquée aux *lignes de produits*, présente des problèmes encore non résolus [IBK11]. Dans le contexte d'une approche de variabilité positive, la définition d'un ordre de fusion associé à l'algorithme de dérivation de produits a été identifiée comme un des points importants à traiter.

Les travaux réalisés lors du stage master recherche de David Suarez [Sua11], en lien direct avec les travaux de thèses, portent principalement sur des études de la composition de machines à état et la transformation de diagrammes de séquence en diagrammes de machines à état. Conjointement à ces travaux, une application d'implémentation de *relationships* a été menée afin d'adresser de manière pragmatique le problème de dérivation de diagrammes comportementaux (tels les diagrammes de séquences). Inclure une notion d'ordre total dans le diagramme de caractéristiques d'une *ligne de produits* de grande taille est bien entendu une tâche complexe ; cette approche est rejetée dans [IBK11], qui considère le passage à l'échelle non réalisable.

L'approche présentée dans cette section n'a pas prétention à résoudre la problématique de composition comportementale, mais plutôt de présenter un cas d'application du langage PLiMoS. La technique repose sur le savoir faire des ingénieurs et leur capacité à établir un ordre global de composition, l'approche prônant l'explicitation de ces relations d'ordre établies, dans l'exemple présent, sur un modèle de variabilité de type modèle de *features*.

13.2 Outillage développé dans le cadre de la thèse

Une exigence utilisateur communément exprimée est d'avoir de l'outillage intégré, au lieu de manipuler des outils hétéroclites. En conséquence, les prototypes outillés réalisés en support de cette thèse sont réalisés dans la plate-forme eclipse, autour de Java et de la surcouche de modélisation EMF [2].

13.2.1 Un panel de développements

Tout d'abord, les modeleurs UML connectés sont ceux d'eclipse, de Rhapsody [6] (via le *plug-in* adapté), et de RSA [7] (dans le cadre de la modélisation comportementale).

Par ailleurs, les éditeurs graphiques spécifiques sont réalisés avec Obeo Designer [11, JB10], un outil commercial intégré à eclipse, orienté modèles (surcouche de GMF) et supportant l'approche multi-points de vues introduite dans la méthodologie ARCADIA [Voi08].

De plus, les générations et transformations autour de PLiMoS$_{specification}$ sont implémentées en Java EMF. La translation dans l'espace technologique de la logique propositionnelle depuis PLiMoS et les modèles de variabilité est réalisée par l'implémentation d'un visiteur et d'algorithmique en Java.

De même, le processus d'évolution incrémentale propose un interfaçage utilisateur via des fenêtres eclipse de type *wizards* avec une algorithmique implémentée en Java. Les fusions de modèles, pour la dérivation de produits sur la base d'une variabilité positive sont réalisées pour partie en Java EMF (modèles de *features*, MyCCM, Ecore), et également en Kermeta [18] et Kompose (Ecore et diagrammes de classes UML).

En effet, tel qu'annoncé dans la section 11.5.1 l'approche utilise pour la composition des *Core Assets* une technique orientée modélisation par aspects nommée Kompose [FBFG07], basée sur le principe d'une fusion systématique d'éléments identifiés comme identiques. De manière similaire, les algorithmes de fusion basé Java EMF se fondent sur la comparaison de leurs signatures [RFG+05].

13.2.2 Quelques précisions sur l'édition graphique

Le choix d'une syntaxe concrète diagrammatique, reste somme toute très pragmatique, permet de fusionner au mieux avec les langages ciblés, que sont les modèles de *features*, *OVM*, *UML* et *ECore* par exemple. Cette dernière est, dans le cadre de PLiMoS, réalisée avec Obeo Designer (modélisation intentionnelle et divers modèles de spécialisation). À noter que l'implémentation d'une syntaxe textuelle serait réalisable mais à ce jour non désirée.

L'outillage de visualisation des *relationships* se doit d'avoir les prédispositions suivantes :

- Un filtrage permettant de se concentrer sur un sous-ensemble d'éléments spécifiques, par ex. des *relationships* affectant une décision spécifique ou déclenchée par un type de décision.
- Une recherche, par nom, par type de relation, etc.
- Une visualisation de l'analyse d'impact, quels sont les variants affectés par telle décision ? Comment ses conséquences sont elles propagées dans les modèles ? Par quelles *relationships* ? Comment et pourquoi tels artefacts sont affectés par telle décision ?

- Une possibilité de navigation dans l'univers large des *relationships* avec la possibilité de faire des zooms avant /arrière.
- Des possibilités de création, *Drag & Drop*, etc.

À titre d'exemple, en suivant le cas illustratif du contrôle anti-aérien introduit dans la section 12.3.1, et en considérant cette fois un EVM de type modèle de *features*, et un IVM de type OVM. Des éditeurs graphiques sont réalisés pour chaque modèle de variabilité et, telles des surcouches, des filtres spécifiques Obeo Designer sont conçus pour par exemple montrer la présence de lien de satisfaction (cf. figure 13.3 - a), le *feature* "2 Pos." est coloré différemment car relié par un lien de satisfaction. Ce type de filtre fournit une compréhension initiale à l'utilisateur.

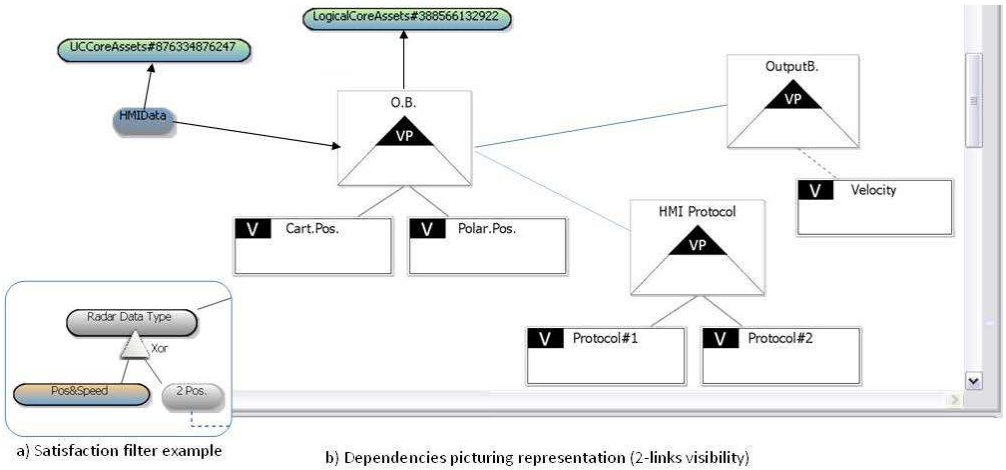


FIGURE 13.3: Modélisation de la variabilité et dépendances PLiMoS - outillage

La face cachée des modèles de variabilité réside dans les interactions transverses entre les éléments. Le point de vue *relationships* PLiMoS permet de se focaliser sur ces dépendances. Par exemple, d'aucun peut choisir un élément constituant un épicentre, les relations sont extraites et calculées, pour être visualisées. A titre illustratif, dans la figure 13.3 - b - l'attention est portée sur l'élément "O.B.", avec une limite de visibilité de deux relations. Sur la figure : "HMIData" fait partie de l'EVM ; "O.B.", "OutputB." et "HMI Protocol" sont des éléments de l'EVM. Pour rappel (cf. section 9.2), l'ensemble des *features* requis (pour un *feature* donné) - *required_features(feature)* - est fourni par une fermeture transitive.

Au delà de l'implémentation des opérateurs décrits dans le chapitre 9, les opérateurs d'édition liés à l'évolution suivant sont implémentés :

- Dupliquer un *feature* ou un variant, ou un point de variation ;
- Dupliquer un *feature* ou un variant, et ses relations (entrantes, sortantes, contraintes) ;
- Insérer un *feature* ou un variant ;
- Modifier un *feature* ou un variant ;
- Séparer un *feature* ou variant (split) : création de n *features* ou variants, avec distribution des fonctionnalités ;
- Suppression d'un *feature* ou d'un variant ;
- Suppression d'un *feature* ou d'un variant et de ses *core assets* liés ;
- Déplacement de la source d'une relation PLiMoS (drag & drop) ;
- Dupliquer une relation PLiMoS existant sur une autre source (avec une *pop-up* intermédiaire) ;
- Supprimer une relation PLiMoS ;

- Fusionner un *feature* mapping (se reporte au parent, les modèles sont des arbres, recherche du premier parent commun)
- Déplacer la cible d'une relation PLiMoS (côté *core assets*) ;
- Dupliquer tout ou partie d'un modèle PLiMoS ;

13.3 Discussion et conclusion

La section propose quelques points de discussion sur la validation, principalement sur l'outillage, pour y présenter des limites et perspectives.

Brièvement, à propos des cas d'études, il est à noter que l'outillage réalisé permet une validation de l'approche par une étude de cas industriel dans l'ingénierie du logiciel et montre l'adéquation et la cohérence de la méthodologie proposée avec le processus de développement en place à Thales. Ce faisant, les autres cas montrent que l'approche peut s'appliquer à une grande variété de domaines d'application de la littérature.

13.3.1 Outillage centré sur PLiMoS

L'outillage ciblé concerne l'édition de PLiMoS et ses diverses opérationnalisations, la section présente certaines limites et évolutions possibles.

Des limitations observées

La principale limitation concerne la gestion de modèles volumineux associés au choix d'une syntaxe concrète principalement diagrammatique et d'un produit tel Obeo Designer. La problématique de représentation graphique et de visualisation d'un grand nombre de données est une problématique à part entière. Dans ce sens, Pleuss *et al.* [PRB11] proposent une étude applicative de différentes techniques de visualisation pour la configuration de produits.

L'utilisation de graphes, et *a fortiori* de modèles, à des fins de représentation de dépendances de *features* fait face à la critique du non passage à l'échelle [BC00]. L'outillage proposé est bien ici qu'un concept de preuve, le souci de l'implémentation et de ses performances n'est pas primordiale à cette étude.

Des perspectives d'évolution

Une "remonté" des résultats dans l'espace propositionnel n'est pas réalisée à ce jour, cependant, une telle gestion permettrait d'obtenir une configuration des modèles de variabilité de type dérivation incrémentale, proposant une diminution des choix possibles à chaque étapes par inférence (les options de sélection devenant : selected, not selected enable, not enabled, valid, invalid, etc.).

Une deuxième perspective fait référence à la génération de PLiMoS et à l'évolution de PLiMoS spécification. Une génération automatique de la syntaxe graphique, c.-à-d. du modèle de description de la syntaxe graphique interprété par Obeo Designer, offrirait une grande plus-value. Des associations par défaut pourraient être envisagées (associations : feature - node, relation - création d'association ou de node avec des ports suivant la cardinalité, etc.).

Enfin, une tout autre perspective concerne le choix et développement d'une syntaxe textuelle, qui offrirait un ajout de "sucre syntaxique" permettant des gains de saisie pour des saisies de contraintes complexes (par ex. *almost* tant de variants, *atleast*...).

13.3.2 Outillage autour de l'évolution

L'outillage concerné par cette section est celui associé au processus de découverte et remonté des nouvelles variabilités, processus permettant une évolution par extension de la *ligne de produits*.

le pourquoi d'un processus semi-automatique

Afin d'apporter une aide à l'utilisateur lors du processus d'évolution de la *ligne de produits*, toute décision pouvant être déduite se doit d'être inférée, et ce, afin de ne pas surcharger d'information l'utilisateur. Cependant, il est assurément impossible et non souhaitable d'atteindre une approche complètement automatisée, l'expertise métier étant à ce jour un point clé d'une organisation pérenne de la *ligne de produits*. Alors qu'une approche semi-automatique permet d'atteindre un plus haut degré de productivité, une approche complètement automatisée tend à compliquer les choses par l'introduction de comportements trop génériques, voire non souhaitables dans certains cas d'application concrets.

Des limitations observées

De la même manière que dans les limitations précédentes se retrouve le problème du passage à l'échelle, l'algorithme étant implémenté en Java EMF sans optimisation particulière, un problème pressenti concerne le temps de traitement en présence de modèles et de bases de *core assets* conséquents.

Par ailleurs, l'algorithme de détection des similitudes se base sur une approche de détection des signatures, pour laquelle il est nécessaire pour chaque nouveau DSL adressé de définir de nouvelles règles. Une méthodes plus générique permettrait de pallier ce problème.

Des perspectives d'évolution

Plusieurs perspectives de travail sur le thème de l'évolution incrémentale semblent dignes d'intérêt. De prime abord, une réflexion autour de l'ajout d'algorithme d'apprentissage permettrait de proposer des solutions d'incorporation des éléments variables suivant un ordre préférentiel. D'autre part, la représentation des choix offerts étant limitée (présentation d'un *wizard* avec des choix multiple), il serait particulièrement appréciable d'ajouter des interfaces de visualisation graphique de la cible de rattachement de la nouvelle variabilité.

Cinquième partie

Conclusion

"I may not have gone where I intended to go, but I think I have ended up where I needed to be."
— Douglas Adams, *The Long Dark Tea-Time of the Soul, 1988*

CHAPITRE 14

Conclusion et perspectives

"Great things are done by a series of small things brought together."

—Vincent van Gogh

DANS le contexte de la thèse, qui propose de tirer parti pleinement de l'*ingénierie dirigée par les modèles* et des *lignes de produits*, l'accent a été mis sur le décomposition, l'organisation et la gestion de la variabilité, tout au long du processus de développement - au travers des niveaux d'abstraction contexte, logique et physique - prenant en considération des préoccupations métier, au travers de diverses ingénieries - système, logiciel et matériel.

Suite à un état de l'art, axé sur la modélisation, celle plus spécifique de la variabilité, et enfin l'évolution dans les *lignes de produits*, l'exposé du développement de la thèse s'est organisé autour de trois parties traitant (i) de la modélisation de l'espace de modélisation des *lignes de produits*, (ii) de l'évolution incrémentale de ces dernières, et (iii) d'une vérification de l'applicabilité de l'approche présentée ci-avant.

Cet ultime chapitre s'organise autour de trois points, la synthèse des contributions proposées par la thèse, une conclusion et l'évocation de perspectives de travail et interrogations persistantes au terme de cette réflexion.

14.1 Synthèse des contributions

L'objectif de la thèse est singulier : maîtriser des incréments d'évolution d'une Ligne de Produits de systèmes complexes. La thèse est que 1^o) une variabilité multidimensionnelle et une identification claire des préoccupations et de leurs interrelations sont requises dans le cadre de *lignes de produits* de systèmes complexes pour en améliorer la compréhension et faciliter l'évolution et que 2^o) les efforts de spécialisation lors de la dérivation d'un produit doivent être capitalisés dans un processus semi-automatisé d'évolution incrémentale des *lignes de produits*.

En résumé, les apports sont tout d'abord d'ordre méthodologie, concernant (i) spécifiquement les *lignes de produits* (avec un choix de cadre conceptuel de décomposition de la variabilité, des apports concernant l'architecture et le processus d'évolution), mais aussi (ii) une vision plus large de la modélisation (avec le choix de la modélisation relationnelle et son couplage avec la vue intentionnelle).

Les contributions de la thèse sont de multiple nature :

1. *Une étude des relations et de leur sémantique et usages dans les approches des lignes de produits* ;
2. *Un cadre conceptuel générique de décomposition multidimensionnelle de la variabilité suivant quatre axes* ;
3. *Une modélisation explicite de l'intentionnalité de l'espace de modélisation des Lignes de Produit* ;
4. *Une modélisation des interdépendances et relations entre les éléments de modèles, et de la sémantique liée à leurs usages* ;
5. *Une structuration de l'espace de modélisation par la définition d'une plate-forme conceptuelle basée sur un patron pour guider et contraindre la dérivation et l'évolution de la Ligne de Produits* ;
6. *Un processus d'évolution incrémental des lignes de produit par extension qui prend son essor dans la dérivation de produits.*

De manière sous-jacente à ces propositions apparaît le langage de modélisation (*Domain Specific Modeling Language - DSML*) nommé PLiMoS permettant une modélisation centrée sur l'aspect relationnel de l'espace de modélisation des *lignes de produits*.

Avec PLiMoS, les *relationships* ne sauraient se réduire ni à leurs retombées techniques, ni à de pures manipulations formelles. L'ajout de sémantique relationnelle aide à la compréhension des dépendances et des interactions entre modèles et éléments de ces derniers, dans l'espace de modélisation. En outre, la séparation de la structure relationnelle et de ses interprétations sémantiques permet un découplage des relations et de ses usages, favorisant de ce fait la perception des activités, et la réutilisation des structures relationnelles dans divers processus. Réifier ces dépendances favorise la communication et le co-développement de modèles pour en renforcer la cohérence. La gestion des relations via un DSML facilite l'outillage et donc la création de vues spécifiques et dédiées pour les différents utilisateurs, en plus d'un support outillé des activités des *lignes de produits* (vérification de cohérence, dérivation, etc.).

Bien que son origine prenne corps après des études et réflexions dans un contexte particulier des *lignes de produits*, le langage PLiMoS ne saurait être limité à ce domaine ; sa généricité le rend efficient pour tout espace de modélisation du *modelware*. Cette problématique de "fédération" de modèles trouve un essor récent dans la communauté de l'IDM, avec la création d'une RFP [15] de l'OMG, ainsi que d'un *workshop* en marge de MODeLS : Semantic Information Modeling for Federation ; ceci constitue une preuve, s'il en faut, de l'importance du problème à résoudre.

14.2 Conclusion

La contribution principale de cette thèse est qu'elle améliore la modélisation des *lignes de produits* et des activités impliquées pour augmenter leur flexibilité et supporter une évolution ciblée.

Les propriétés défendues par cette thèse pour une mise en œuvre d'une *ligne de produits* dirigée par les modèles sont identifiées et définies comme formant trois points clés distincts :

1. Explicitation : explicitation de l'espace de modélisation, des relations et intentions de collaborations entre modèles, facilitant de ce fait une compréhension du processus ;
2. Structuration : structuration de la LdP, tant sur le plan de la maîtrise des modèles que sur l'architecture ;
3. Flexibilité d'évolution : le processus d'évolution doit être clarifié et varié, permettant une aide concrète à l'évolution.

Les solutions proposées s'organisent autour d'un langage de modélisation relationnelle nommé PLiMoS et concernent l'architecture et le processus de gestion de l'évolution. Le langage PLiMoS

apporte les bénéfices suivants :

- il permet de sortir de l'abstinence sémantique chronique des relations dans les divers langages de variabilité ;
- il facilite la gestion de modèles hétérogènes ;
- il renforce la passerelle modélisation du système et modélisation du processus de développement.

Les solutions techniques, à savoir le langage PLiMoS, le motif d'architecture (dans notre cas ABCDE) définissant la plate-forme conceptuelle de la LdP, ainsi que le processus outillé d'évolution, rencontrent une bonne adaptation et adoption dans les cas des applications traitées.

En résumé, la structuration et l'organisation de la modélisation et du processus de la *ligne de produits* sont mises en avant par l'approche afin d'éviter l'évolution ne vire à l'involution.

14.3 Perspectives

Au delà des considérations techniques autour de l'outillage, et dont les pistes de réalisation ont été évoquées dans le chapitre 13, la thèse laisse en suspend des pistes et lève des interrogations.

14.3.1 Concernant PLiMoS et les relations sémantiques dans le cadre des *lignes de produits*

La granularité des relations pouvant être diverse, et leur nombre très imposant, il apparait dès lors un besoin, comme pour tout processus fastidieux et rébarbatif, de faciliter l'automatisation de la gestion des relations, qui passe par leur découverte, et leur maintien en cohérence. L'émergence et la découverte d'interactions entre propriétés est une problématique à part entière identifiée par la communauté des *lignes de produits*.

En outre, l'analyse et le développement systématique de métriques de haut niveau (pas uniquement de comptage) sur la base des relations permettrait de faciliter le développement et la maintenance des *lignes de produits*, et, à terme, de faire remonter des motifs de modélisation adaptés.

14.3.2 Sur un plan architectural

Les observations ci après concernent l'architecture et le développement à Thales. Pour aller plus loin que l'exposé de cette thèse, des actions sont à mener :

- concernant le patron d'architecture, formaliser la démarche d'alimentation de la plate-forme conceptuelle procurée par le patron d'architecture ABCDE depuis le niveau de modélisation du contexte (*Uses-cases* et *Goal-cases* principalement) ;
- concernant le déploiement sur l'ensemble du processus de développement, analyser finement les patrons applicables à l'ingénierie système et la translation de la variabilité système à l'ingénierie logicielle ;
- dans une perspective de généralisation de l'approche, définir des règles de contextualisation de l'application des patrons d'architecture pour une adaptation au domaine métier et aux caractéristiques des logiciels ciblés.

14.3.3 Sur la thématique de l'évolution

L'évolution est une occurrence naturelle du développement logiciel, et elle engendre une part irrévocable de gestion des *lignes de produits* et de leur architecture. En raison des dépendances

intrinsèques entre les *core assets* de la LdP, des modifications sur un *asset* peuvent impacter et nécessiter des changements sur un ou plusieurs autres.

Malgré les dispositions de l'architecture à être relativement ouverte (par l'approche de plate-forme conceptuelle basée sur les patrons), ces modifications peuvent engendrer une évolution marquée de la *ligne de produits*. Face à cette sempiternelle évolution, la thèse s'est attachée à traiter l'aspect évolution par extension, caractéristique d'une LdP réactive. Cet aspect n'est cependant pas unique et une perspective intéressante concerne la gestion de l'évolution corrective de la *ligne de produits*, c.-à-d. la prise en compte de modification de la LdP et de ses composants tout au long de son cycle de vie, en y intégrant des problématiques de version et d'historique.

Cette problématique n'est pas triviale, et amène à s'interroger sur les aspects spécifiques aux *lignes de produits* réactives ayant un long cycle de vie. En prenant comme hypothèse, somme toute raisonnable, qu'une évolution corrective des composants ne concerne que les développements en cours (et éventuellement pas tous suivant leur état d'avancement) et futurs, comment préserver l'intégrité de la LdP lorsque une correction est apportée à un produit précédemment déployé ?, ces modifications doivent-elles intégrer la *ligne de produits* ?, une multitude de branches n'a-t-elle pas été créée ? Les préoccupations de gestion de configuration et de version, qui constituent déjà un problème épineux des développements logiciels traditionnels, se voient augmentées en complexité par l'ajout de la dimension de variabilité des *lignes de produits*.

Des perspectives de travaux sont brièvement listées ci après :

- analyse d'impact et gestion du risque (conservation d'un historique des décisions, des causes et stratégies de réponses existantes, identification de catégories de risques en terme d'impact) ;
- augmentation du degré d'automatisation (génération intégrale de la syntaxe graphique diagrammatique et de l'outillage, co-évolution et génération des transformations) ;
- encapsulation de la modélisation des relations et de la variabilité, ainsi que des modèles PLiMoS ;
- travail collaboratif à prendre en compte (gestion de configuration) ;
- examen de l'extension de cette modélisation à de multiples espaces techniques (et divers outils, tel Rhapsody, Doors Treck, etc.) mettant en exergue des problématiques de chaine d'outils et d'écosystèmes.

14.3.4 Le langage PLiMoS et l'évolution

Un modèle PLiMoS de modélisation de l'espace de modélisation est une auto-organisation qui se forme par l'ouverture et l'activité du système modélisé dans les contextes avec lesquels il co-évolue, qu'il transforme et qui le transforment.

Les besoins peuvent aller au delà de l'évolution du modèle en impactant le niveau méta également. Or, dans l'état actuel des choses, l'évolution du langage et la co-évolution des modèles supportés $PLiMoS_{specialisation}$ ne concernent quasiment que la gestion par extension, soit un pan minime et relativement simple du suivi de l'évolution des relations.

De par sa construction mettant en avant définition et génération, le langage manque de souplesse de modélisation. Il serait intéressant d'avoir des facilités de modélisation inverse, qui par exemple partiraient d'une représentation graphique pour ensuite venir se voir adjoindre de la sémantique *a posteriori* ; une telle adjonction de sens et de contraintes sémantiques (voir syntaxiques) résulte en la définition d'un nouveau langage ou la modification d'un langage existant.

Enfin, couplé à la perspective de la remontée d'information évoquée dans la section 14.3.1, la proposition permettrait une sérendipité systématique, c.-à-d. une découverte de connaissance et une considération immédiate de celle-ci par le langage PLiMoS.

14.3.5 Un lien plus formel avec la modélisation du processus de développement

Chercheurs et praticiens ont reconnu depuis longtemps que la décomposition des efforts de communication et de coordination constitue un problème majeur dans le développement logiciel [CKI88]. Une des raisons de ce problème réside dans le nombre important de dépendances impliquées dans un développement : dépendances entre les différents artefacts logiciel, et dépendances entre les diverses activités du processus de développement.

D'une part, les dépendances techniques, les séparations des préoccupations, créent des dépendances sociales, qui présentent un intérêt à être modélisées. D'autre part, la problématique de maintenance des relations et de l'attribution des responsabilités à des rôles (acteurs du processus) est centrale. La modélisation passe alors d'une attitude d'analyste à une attitude de concepteur, plus attentif au processus (la modélisation) qu'au résultat (le modèle).

Un premier rapprochement est réalisé entre la modélisation relationnelle et la modélisation du processus de développement, par l'approche basée modèles et abstractions, et par l'intention couvrant toute modélisation.

Il apparaît intéressant d'approfondir ce lien et éventuellement d'envisager un couplage processus - PLiMoS, permettant si ce n'est une génération de sous ensemble de modèles, une vérification accrue de la cohérence de l'ensemble. Une vérification et un suivi de l'adéquation de la modélisation du processus et de l'espace de modélisation suivant les relations intentionnelles permettraient d'éviter toutes déviations dans les usages des modèles.

14.3.6 Vers une modélisation relationnelle plus avancée

Les réflexions autour du besoin de connaissance relationnelle dans les *lignes de produits* dirigées par les modèles amènent à considérer plus largement le problème de modélisation relationnelle et à l'étendre à toute modélisation. Il apparaît légitime de s'interroger sur la suffisance de la place des relations dans le cadre de modélisation du MOF, exprimant les limites d'expressivité et de spécialisation des relations dans un métamodèle.

Face à l'étroitesse de ce cadre, et de la place restreinte accordée à la notion de relation, des interrogations sont levées : Quid d'une augmentation du M3 ?

Annexes

ANNEXE A

Traductions diverses du document

Cette annexe présente diverses traductions de l'anglais vers le français de citations représentant pour la plupart des définitions.

A.1 Modélisation et Ingénierie Dirigée par les Modèles

Citation 2.1.1, p 15 : "[...] l'abstraction est la projection d'une représentation d'un problème sur un autre, qui préserve certaines propriétés désirables et qui réduit la complexité." [GW92]

Citations de la table 2.1, p 16 :

- [MoTAIL65] : "Pour un observateur B, un objet A* est un modèle d'un objet A *ssi* B peut utiliser A* pour répondre aux questions qui l'intéressent sur A."
- [BRJ99] : "Un modèle est une simplification de la réalité [...] telle qu'elle améliore la compréhension du système en développement."
- [BG01] : "Un modèle est une simplification d'un système construit avec un but particulier en tête. Le modèle doit pouvoir répondre aux questions en lieu et place du système actuel."
- [Jac02] : "Ici le mot modèle représente une part de l'espace mémoire machine ou de la base de données qui conserve, de manière plus ou moins synchronisée, une correspondance avec une part du domaine du problème. Le modèle peut par la suite se substituer au domaine du problème, fournissant des informations aux machines qui ne peuvent pas être aisément obtenues à partir du domaine du problème lui-même à tout instant."
- [Lud03] : "Les modèles participent aux développements d'artefacts en fournissant des informations sur les conséquences de leur construction avant leur réalisation effective."
- [OMG03b] : "Un modèle d'un système est une description ou spécification de ce système et de son environnement établi dans un but précis."
- [Sei03] : "Un modèle est un ensemble d'affirmations sur des systèmes en analyse."
- [Sel03] : "Concevoir des modèles permet de réduire les risques par delà le fait qu'ils favorisent la compréhension, d'une part, d'un problème complexe, et d'autre part, de ces solutions potentielles avant d'engager les efforts et dépenses nécessaires à une implémentation complète."
- [Bro04] : "Les modèles fournissent des abstractions de systèmes physiques qui permettent aux ingénieurs de réfléchir sur le système en ignorant des détails superflus et en se concentrant uniquement sur ceux pertinents."

- [Küh06] : “Un modèle est une abstraction d'un système (réel ou basé langage), permettant de réaliser des prédictions ou des inférences à partir de celui-ci.”

Citation 2.2, p 17 : “Nous savons qu'un programme doit être correct et nous pouvons l'étudier de ce point de vue uniquement ; nous savons également qu'il devrait être efficace et nous pouvons étudier son efficacité d'autre part. Suivant les modes nous pouvons nous demander, s'il y a lieu : Pourquoi le programme est désirable. Mais l'on n'y gagne rien, -bien au contraire !- à traiter ces divers aspects simultanément. C'est ce que j'ai parfois nommé “la séparation des préoccupations”, qui, même si elle n'est pas parfaitement réalisable, est pour le moment la seule technique disponible pour organiser ses idées, d'aussi loin que je puisse connaître. C'est ce à quoi je fais référence par “focaliser l'attention de quelqu'un sur un certain aspect” : cela ne signifie pas ignorer les autres aspects, c'est juste considérer le fait que depuis le point de vue de tel aspect, tel autre est sans rapport. C'est suivre un et plusieurs chemins de pensée à la fois.”

Citation 2.2.1, p 19 : “un ensemble de sous-systèmes et technologies qui fournissent un ensemble cohérent de fonctionnalité au travers d'interfaces et de patrons d'usages spécifiés, que toute application supportée par la plate-forme peut utiliser sans se préoccuper des détails d'implémentation des fonctionnalités.”

Citation 2.3.1, p 22 : “Le métamodèle est il un modèle d'un modèle ? [...] La réponse est entre les deux, ou mieux dit, quelqu'un peut dire OUI ou NON, en fonction de l'interprétation du mot modèle.”

Citation 2.3.2, p 23 : “**Metamodelisation Stricte** : Dans une architecture de modélisation à n niveaux, M_0, M_1, ..., M_{n-1}, chaque élément de modèle d'un niveau de modélisation M_m doit être une “instance-de” exactement un élément d'un modèle de niveau M_{m-1}, pour chaque $0 \leq m < n - 1$, et toute relation autre que “instance-de” entre deux éléments X et Y implique que le niveau de X est identique à celui de Y.”

Les définitions de mégamodèle (section 2.3.3, page 24) :

- [BJV04] : un mégamodèle est un modèle ayant d'autre modèles comme éléments.
- [Fav05b] : “l'idée derrière un mégamodèle est de définir un ensemble d'entités et de relations qui sont nécéssaires à modéliser certains aspects de l'IDM.
- [SME09] : un macromodèle consiste en des éléments représentant des modèles et des liens représentant des relations entre ces modèles faisant fi des détails internes.

Traduction de la figure 2.6, p 27 :

- Type différent, description : “X et Y ont des intentions totalement différentes. Cela dénote habituellement un saut sémantique entre deux points de vues.”
- Type partage, description : “X et Y partagent des intentions. X et Y peuvent être partiellement représentés l'un par l'autre. La représentation est partielle et étendue.”
- Type sous, description : “L'intention de X fait partie de l'intention de Y. Tout élément de X a du sens dans le contexte de Y. Y peut être partiellement représenté par X.”
- Type identique, description : “X et Y partagent la même intention. Ils peuvent se représenter l'un l'autre. Cela dénote habituellement un saut linguistique.”
- Type sur, description : “X couvre l'intention de Y ; X peut représenter Y, mais X a des propriétés additionnelles. C'est une représentation étendue.”

Citation 2.4, p 28 : “[...] une transformation est une génération automatique d'un modèle cible depuis un modèle source, conformément à une définition de transformation. Une définition de transformation est un ensemble de règles de transformations qui, dans leur globalité, décrivent comment le modèle dans le langage source peut être transformé en un modèle dans le langage

cible. Une règle de transformation est une description de comment une ou plusieurs constructions du langage source peuvent être transformées en une ou plusieurs constructions du langage cible."

Citation 2.5.1, p 30 :

- "Le degré auquel une relation peut être établie entre deux ou davantage de produits du processus de développement, plus spécifiquement des produits liés entre eux par des relations prédécesseur/successeur ou maître/esclave [...].
- Le degré auquel chaque élément d'un développement logiciel établit ses raisons d'existence [...]."

A.2 Modélisation de la variabilité

Citation 3, p 35 : "Nous considérons un ensemble de programmes comme constituant une famille s'ils ont tant en commun qu'il est avantageux de s'intéresser à leurs aspects communs avant de regarder leurs différences. Cette définition relativement pragmatique ne nous dit pas ce qui paie, mais nous renseigne sur les motivations de réaliser des familles de programmes. Nous voulons exploiter les commonalités ... et réduire ... les coûts."

Citations de la table 3.1, p 37 :

- [WL99] : "une hypothèse sur comment les membres d'une même famille diffèrent entre eux."
- [BC05] : "la variabilité signifie la capacité d'un "core assets" de s'adapter à des usages dans les différents contextes produits qui sont dans le périmètre de la *ligne de produits*."
- [PBL05] : "la variabilité qui est modélisée pour permettre le développement d'applications spécifiques en réutilisant des artefacts prédéfinis et ajustables."
- [SvGB05] : "la variabilité est la capacité d'un système à être efficacement étendu, adapté ou configuré pour un usage donné dans un contexte particulier."

Citation 3.2, p 39 : "[...] un ensemble de décisions qui sont suffisantes pour distinguer les membres d'une famille de produits d'ingénierie de l'application, et de guider l'adaptation d'un produit d'ingénierie de l'application résultante."

Citations de la table 3.2, p 45 :

- FODA [KCH+90] (Kang et al.) : "Un aspect visible de l'utilisateur, proéminent et distinct, une qualité ou une caractéristique d'un ou des systèmes logiciels".
- FORM [KKL+98] (Kang et al.) : "des abstractions fonctionnelles identifiables et distinctes qui doivent être implémentées, testées, et maintenues."
- Generative Programming [CE00] : "Propriété d'un concept du domaine, qui est pertinent pour certaines parties-prenantes et utilisé pour discriminer des instances de concepts."
- IEEE [oEE90] : "Une caractéristique logicielle spécifiée ou impliquée par les exigences documentées (par ex. des fonctionalités, des performances, des attributs, ou des contraintes de conception)".
- Riebish et al. [Rie03] : "Un *feature* représente un aspect ayant une valeur pour l'utilisateur."
- Bosch [Bos00] : "une unité logique de comportement qui est spécifiée par un ensemble d'exigences fonctionnelles et de qualité."
- Chen et al. [CZZM05] : "une caractéristique d'un produit vue du point de vue utilisateur, qui consiste essentiellement en un ensemble cohérent d'exigences individuelles."
- Batory [Bat05b] : "une élaboration ou augmentation d'une ou plusieurs entités qui introduisent un nouveau service, une nouvelle capacité ou une nouvelle relation."
- Batory et al. [Bat05a] : "un incrément de fonctionalité d'un produit."

- Apel et al. [ALMK08] : “une structure qui étend et modifie la structure d'un programme donné dans le but de satisfaire des exigences d'une des parties prenantes, pour implémenter et encapsuler une décision de conception, et offrir une option de configuration.”

A.3 *Lignes de produits* et évolution

Citation 4.1.1, p 63 : “Une LdP logicielle est un ensemble de systèmes à logiciel prépondérant partageant un ensemble commun et organisé de *features* qui satisfont les besoins spécifiques d'un segment de marché donné, ou une mission et qui sont développés à partir d'un ensemble commun de *core assets* d'une manière prescrite.”

Citation 4.1.1, p 63 : “Une LdP logicielle consiste en une architecture de LdP et un ensemble de composants réutilisables qui sont conçus pour incorporer l'architecture de la LdP. En complément, la LdP est constituée des produits qui ont été développés en utilisant les *assets* réutilisables évoqués précédemment.”

Citation B.3.1, p 214 : “L'architecturelogicielle d'un programme ou d'un système logicel est la structure ou les structures de ce système, ce qui comprend les éléments logiciel, les propriétés externes et visibles de ces éléments, et les relations entre eux.”

Citation B.3.1, p 214 : “une architecture logicielle consiste en des composants, des connecteurs et des contraintes additionnelles . . . ”

Citation B.3.1, p 215 : “Chaque patron est une règle à trois intrants, qui exprime une relation entre un certain contexte, un problème, et une solution.”

Citation B.3.1, p 216 : “Un patron architectural exprime un schéma structurel fondamental d'organisation d'un système logiciel. Il fournit un ensemble de sous-systèmes prédéfinis, spécifie leurs responsabilités, et inclut des règles et guides d'organisation des relations entre eux.”

Citation B.3.1, p 216 : “Les patrons fournissent un vocabulaire commun et une compréhension commune de principes de conception.”

Citation B.3.1, p 216 : “Un patron de systèmes d'une architecture logicielle est une collection de patrons d'architecture logicielle, avec des guides pour leur implémentation, leur composition et leur utilisation pratique lors d'un développement logiciel.”

Citation B.3.1, p 217 : “Un style architectural définit une famille de systèmes logiciels en termes d'organisation structurelle. Un style architectural exprime des composants et des relations entre eux, et des contraintes de leur application, mais également les règles de composition et de conception associées utiles à leur construction.”

Citation 4.2.3, p 66 : “[. . .] une architecture de base qui capture les détails de conception de haut niveau pour les produits d'une LdP, en incluant les points de variation, les variants documentés dans le modèle de variabilité.”

Citation 4.2.3, p 66 : “L'architecture de la LdP est un membre important et premier de la collection des *core assets*. (. . .) L'architecture définit un ensemble de composants logiciels (. . .) qui peuplent la base des *core assets*. L'architecture des LdP - combinée au plan de production - fournit la description (. . .) de la manière de construire les produits à partir des *core assets*.”

A.4 Considérations architecturales et précision du périmètre de la LdP

Citation 10, p 140 : "[...] une équipe d'ingénieurs brillants peut être en mesure d'effectuer un bon travail avec une piètre architecture logicielle. D'un autrre côté, une architecture logicielle "parfaite" peut amener à des résultats inacceptables, mise dans les mains d'ingénieurs inexpérimentés qui échouent à comprendre la logique sous-jacente à l'architecture logicielle."

ANNEXE B

Compléments à l'État de l'Art

Cette annexe présente un complément à l'état de l'art du tapuscrit de thèse. Les informations qui y sont consignées sont de diverses natures et représentent un approfondissement de l'état de l'art. Ainsi y figurent, entre autres, les données intermédiaires sur l'étude des relations dans les modèles de variabilité, ainsi qu'une présentation plus complète de l'architecture des *lignes de produits*. Les sections renseignent sur la thématique étendue : Modélisation et Ingénierie Dirigée par les Modèles, Modélisation de la variabilité, Lignes de produits et évolution.

B.1 Modélisation et Ingénierie Dirigée par les Modèles

Les axiomes de composition des intentions évoqués page 26 suivent les règles définies par Muller *et al.* [MFBC10]. La figure B.1 propose un rappel de ces règles.

B.2 Modélisation de la variabilité

Les données de l'étude bibliographique sur les relations dans les modèles de variabilité décrite en section 3.4.2, page 3.4.2, sont présentées en détails dans la suite de la section.

Le tableau B.1 regroupe les données brutes issues de l'étude. Y sont listées : les publications avec leur référence, l'année de publication, le type de publication, les types de relations (catégories : *Raffinement* - *R*, Relations de raffinement hiérarchique ; *Config.* - *C*, Dépendances de configuration ; *Usage* - *U*, Dépendances d'usages spécifiques ; *inter* -*i*, Dépendances inter-préoccupations), ainsi que le degré de formalisation *F* (langage naturel sous forme de texte, MétaModèle, Formalisme Mathématique).

Tableau B.1: Distribution des travaux de l'étude.

Publication	*année*	*type*	*Raf.*	*Config.*	*Usage*	*inter*	*F*
FODA [KCH+90]	1990	rapport technique	1	1	0	0	texte
Griss *et al.* [GFA98]	1998	conférence	1	1	0	0	texte
FORM [KKL+98]	1998	journal	1	1	1	1	texte
Czarnecki et Eisenecker [CE00]	2000	ouvrage	1	1	0	1	texte

Publication suite	*année suite*	*type suite*	*Raf. suite*	*Config. suite*	*Usage suite*	*inter suite*	*F suite*
Fey *et al.* [FFB02]	2002	conférence (SPLC)	1	1	1	0	MM
Riebish [Rie03]	2003	workshop	1	1	1	0	texte
Streitferdt *et al.* [SRP03]	2003	conférence	0	1	1	0	MM
Riebish *et al.* [Rie04]	2004	workshop	1	1	1	0	texte
Mei *et al.* [MZZ06]	2006	journal	1	1	1	0	FM,MM
Ferber *et al.* [FHS02]	2002	conférence (SPLC)	1	1	1	1	texte
Lee et Kang [LK04]	2004	journal	1	1	1	0	texte
Jaring et Bosch [JB04b]	2004	conférence (SPLC)	1	1	0	0	FM
Jaring et Bosch [JB04a]	2004	conférence	1	1	0	0	FM
Vranić [Vra04]	2004	conférence	1	1	0	0	FM
Zhu *et al.* [ZLZZ06]	2006	journal	1	1	1	1	FM
Ye et Liu [YL05]	2005	journal	1	1	0	0	MM
Filho *et al.* [SFR06]	2006	conférence (SPLC)	0	1	1	0	texte
Cho *et al.* [CLK08]	2008	conférence (SPLC)	1	1	1	0	texte
Luo *et al.* [DS09]	2009	conférence	1	1	1	1	FM
Botterweck *et al.* [BTN+08]	2008	conférence (SPLC)	0	1	1	1	MM
Zhao *et al.* [ZZM08]	2008	journal	1	1	1	0	MM
Cawley *et al.* [CTBN09]	2009	workshop	0	1	1	1	texte
Choi *et al.* [CLLK09]	2009	workshop	1	1	1	0	texte
Sametinger et Riebisch [SR02]	2002	workshop	0	0	0	1	texte
Czarnecki *et al.* [CA05]	2005	conférence	1	1	0	1	texte
Heidenreich *et al.* [HKW08]	2008	workshop	1	1	0	1	MM
Reiser et Weber [RW07]	2007	conférence	1	1	1	1	MM
Metzger *et al.* [MPH+07]	2007	conférence	1	1	0	1	FM
Classen*et al.* [CHS+10]	2010	conférence	1	1	0	1	FM
Czarnecki *et al.* [CSW08]	2008	conférence (SPLC)	1	1	1	0	FM
Bošković *et al.* [BMB+11]	2011	conférence	0	1	0	1	texte
Hartmann et Trew [HT08]	2008	conférence (SPLC)	0	1	0	1	texte
Lee et Kang [LK10]	2010	conférence (SPLC)	1	1	1	1	FM

Publication suite	*année suite*	*type suite*	*Raf. suite*	*Config. suite*	*Usage suite*	*inter suite*	*F suite*
Sinnema *et al.* [SDNB04]	2004	conférence (SPLC)	0	1	0	1	MM
Deelstra *et al.* [DSB09]	2009	journal	0	1	0	1	MM
Pohl *et al.* [PBL05]	2005	ouvrage	0	1	0	1	MM
Thiel *et al.* [TH02]	2002	conférence (SPLC)	0	1	0	0	MM
Morin *et al.* [MPL+09]	2009	conférence	0	1	0	0	MM
Asikaisen *et al.* [ASM04]	2004	journal	0	1	0	0	MM
TVL [CBH11]	2011	journal	1	1	1	0	FM
DOPLER [DGR11]	2011	journal	1	1	0	0	MM
Mazo *et al.* [MSD+12]	2012	journal	1	1	1	1	MM
Dhungana *et al.* [DSB+11]	2011	conférence (SPLC)	0	1	0	1	texte
Lamb *et al.* [LJZ11]	2011	workshop	0	1	1	1	FM

Le tableau B.2 présente les types de publications.

Tableau B.2: Distribution des travaux de l'étude par catégorie de publication.

Type de publication	*Nb de travaux*
Ouvrage	2
Rapport technique	1
Journal	11
Conférence (GPCE, ECMR, SPLC, ECBS, ICSR, SERP, VaMoS, RE, MoDELS, Net.ObjectDays, ICCET, SDL)	23 (SPLC - 12)
Worskhops (SPLC, ICSE, PFE, ECOOP)	7

Le tableau B.3 présente la distribution des travaux de l'étude, dans le temps et par formalisme.

Tableau B.3: Distribution des travaux de l'étude dans le temps et par formalisme.

Année	*Tot./y*	*Text*	*MM*	*MF*
1990	1	1	0	0
1991	0	0	0	0
1992	0	0	0	0
1993	0	0	0	0
1994	0	0	0	0
1995	0	0	0	0
1996	0	0	0	0
1997	0	0	0	0
1998	2	2	0	0
1999	0	0	0	0
2000	1	1	0	0

Année	*Tot./y*	*Text*	*MM*	*MF*
suite	*suite*	*suite*	*suite*	*suite*
2001	0	0	0	0
2002	4	2	2	0
2003	2	1	1	0
2004	7	2	2	3
2005	3	1	2	0
2006	3	1	1	2
2007	2	0	1	1
2008	6	2	3	1
2009	5	2	2	1
2010	3	0	1	2
2011	4	2	0	2
2012	1	0	1	0

Le tableau B.4 présente la distribution des travaux de l'étude, dans le temps et par catégorie.

Tableau B.4: Distribution des travaux de l'étude dans le temps par catégories.

Année	*# R*	*#C*	*#U*	*#i*
1990	1	1	0	0
1991	0	0	0	0
1992	0	0	0	0
1993	0	0	0	0
1994	0	0	0	0
1995	0	0	0	0
1996	0	0	0	0
1997	0	0	0	0
1998	2	2	0	1
1999	0	0	0	0
2000	1	1	0	1
2001	0	0	0	0
2002	3	4	3	3
2003	1	2	3	0
2004	5	7	2	1
2005	2	3	0	2
2006	2	3	3	1
2007	2	2	1	2
2008	4	5	4	3
2009	2	5	3	3
2010	3	3	1	2
2011	1	4	2	3
2012	1	1	1	1

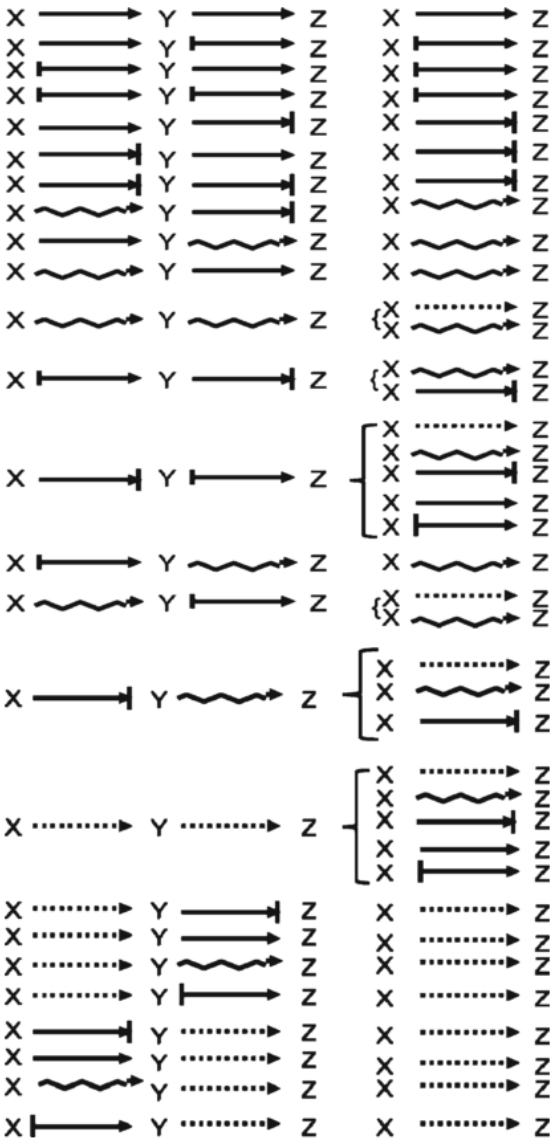


FIGURE B.1: Règles de composition des relations intentionnelles [MFBC10]

B.3 *Lignes de produits* et évolution

Des compléments d'informations concernent l'architecture, ainsi que l'évolution de la *ligne de produits* (illustration de la taxonomie de l'évolution).

B.3.1 Ingénierie du domaine et architecture

Avec la croissance des systèmes logiciels, il est devenu important de planifier et de structurer ces systèmes. La problématique fut identifiée vers la fin des années 1960, début 1970 [McI68, Par72]. Cependant, ce n'est pas avant les années 1990 que la tendance de l'architecture a réellement commencé (Shaw et Garlan [SG96]). Selon Kazman [Kazar], l'architecture est cruciale aux développements logiciels en cela qu'elle (i) favorise la communication entre les parties-prenantes, (ii) réalise une remontée des décisions dans le processus de développement, et (iii) fournit une abstraction à la conception.

Architecture est un terme relativement populaire dans la communauté logicielle et son usage peut varier en fonction du contexte. Il désigne des composants logiciels, la structure de l'unité centrale, la structure organisationnelle d'un système d'information. Il existe également plusieurs interprétations et définitions de ce terme dans le contexte des composants logiciels.

Définition de l'architecture

De nombreuses définitions de la notion d'architecture parsèment la littérature[1] et il n'est pas aisé d'en synthétiser une unifiante.

La première définition de Bass *et al.* [BCK03], souligne l'éventuelle multiplicité des structures de description de l'architecture d'un système. Elle évoque la notion d'organisation et de structure avec une certaine abstraction.

> *"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."*
>
> Bass *et al.* [BCK03](traduction p. 206)

Une seconde définition issue du même ouvrage [BCK03] est plus pragmatique et fait référence à des composants, des connecteurs et des contraintes additionnelles.

> *"a software architecture consists of components, connectors and additional constraints"*
>
> Bass *et al.* [BCK03](traduction p. 206)

D'autre part, la définition de Kruchten [Kru03], par la suite explorée par [JB05] considère l'architecture comme un ensemble de décisions (sur l'organisation, la sélection des éléments, leur composition, et la sélection d'un style architectural guidant les précédentes décisions).

En résumé, l'architecture logicielle peut être comprise d'au moins deux manières différentes : (i) comme un ensemble de décisions (architecturales) de conception, ou (ii) comme la structure résultante de ces décisions. Dans cette thèse, la notion d'architecture s'apparente à la seconde définition. L'essence abstraite de l'architecture logicielle se distingue de la réalisation matérielle représentée dans le programme final.

[1] Le Software Engineering Institute - SEI - (http://www.sei.cmu.edu/architecture/published_definitions.html) en dénombre vingt-sept.

Une description d'architecture logicielle possède quatre rôles centraux dans un projet informatique [SG96]. Elle facilite la compréhension de la structure d'un système. Elle permet son analyse. En tant que cadre pour la construction de l'application, elle sert de base pour la génération de code et l'identification des opportunités de réutilisation. Enfin, elle définit les invariants du système et sert donc de pivot pour son évolution.

Conception de l'architecture

La conception de l'architecture est une activité antérieure à toute implémentation et son objectif est d'assurer la structuration de l'ensemble du système logiciel. Dans la pratique sont fréquemment observées des conceptions d'architecture qualifiée de *ad hoc*, idiosyncratique et informelle. Il en résulte une architecture parfois peu comprise des développeurs, des choix architecturaux ne pouvant pas être analysés en cohérence et en complétude. Enfin, et non des moindres, les choix de conceptions initiaux se voient rapidement disparaître quand le système évolue.

Comme le présente Shaw et Garlan [SG96], de nombreux patrons et styles architecturaux peuvent être employés à cette étape de conception. Notons que, poussé à son paroxisme comme dans l'approche de Ackerman et Gonzales [AG10], le patron se mue en un paradigme de programmation.

Il est souvent mis en avant l'importance de la créativité dans les développements, notamment concernant les systèmes technologiquements innovants. Cependant, une créativité non gérée et ciblée peut apparaitre au détriment du projet. L'objectif est d'autoriser une créativité avec un but, une créativité permettant d'atteindre les objectifs fixés. Les motifs (ou patrons) d'architecture sont utilisés pour restreindre l'espace de la solution de manière collaborative et disciplinée.

Les patrons d'architecture Qu'est-ce qu'un Patron ? Comme le rappel Buschmann *et al.* [BMR+96], dans un domaine donné, pour résoudre un problème particulier, les experts se ramènent souvent à un problème similaire qu'ils ont déjà résolu, et résolvent ce nouveau problème par translation (réutilisation de l'essence de sa solution). Ce genre de comportement chez les experts, c.-à-d. la pensée en termes de paires problème-solution par paires, est commun à de nombreux domaines distincts, tels que l'architecture [Ale79], l'économie [Etz64] et le génie logiciel [BMR+96, GHJV95].

> *"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution."*
>
> Christopher Alexander [Ale79, p247](traduction p. 206)

Le concept de patron a vu le jour avec Christopher Alexander, dans ses travaux relatifs à l'architecture des bâtiments [Ale79], pour ensuite se voir transposé dans le domaine logiciel. De nombreux termes, concepts et principes de l'architecture génie civil se transposent aisément dans le monde logiciel. Il y a cependant de grandes différences et la métaphore architecturale et urbanistique ne doit pas être prise au pied de la lettre. La forme logique d'un programme doit être créée par l'architecte du logiciel, ou par le programmeur. Dans cette construction abstraite logicielle, les briques et artefacts de base sont immatériels.

Les patrons (ou motifs) d'architecture expriment des schémas d'organisation structurelle fondamentaux pour les systèmes logiciels. Ils fournissent un découpage initial en sous-systèmes, chacun ayant des responsabilités spécifiées et bornées, ainsi que la description de leurs interactions :

> *"An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsabilities, and includes rules and guidelines for organizing the relationships between them."*
>
> Buschmann *et al.* [BMR+96](traduction p. ??)

Les patrons décrivent une vision d'ensemble de la conception du système logiciel, permettant une meilleure communication avec des (futurs) partenaires ainsi qu'un contrôle de cohérence. Lors de la modification ou de l'extension de l'architecture, ces gardes-fou permettent de préserver l'intention initiale.

À un niveau d'abstraction inférieur, les patrons d'implémentation (*Design patterns* [GHJV95]) servent d'entité réutilisable *(building bloc)* pour construire des structures plus complexes.

> *"Patterns provide a common vocabulary and understanding for design principles."*
>
> GoF [GHJV95](traduction p. 206)

Il apparaît, à ce moment, nécessaire d'insister sur la distinction entre les patrons de réalisation (ou de concéption), et les patrons de type mannageriaux. Les patrons de réalisation précisent la structure de la solution, et ont conséquemment un impact notoire et visible sur le produit logiciel final. À l'inverse, les patrons de management sont mis en œuvre dans le processus amenant à la conception du logiciel, et n'apparaissent sous aucune forme dans la solution finale. L'ouvrage de référence des LdPs de Clements et Northrop [CN01] par exemple fait la part belle à ces patrons organisationnels et de processus.

Les patrons n'existent pas isolément, de nombreuses interdépendances peuvent être révélées. Les patrons sont tissés dans un "pattern system" [BMR+96], qui décrit comment les patrons sont connectés et comment ils se complètent entre eux. Un système de patron est défini par Buschmann *et al.* comme :

> *"A pattern system for software architecture is a collection of patterns for software architecture, together with guidelines for their implementation, combination and practical use in software development."*
>
> Buschmann *et al.* [BMR+96](traduction p. 206)

Des exemples de Patrons

De nombreux patrons d'architecture logiciels ont été recensés [BMR+96], fruits de l'expérience de développeurs. Néanmoins, beaucoup d'entre eux sont dédiés à des applications spécifiques, citons pour exemple : le "Model-View-Controller" pour des applications orientées Interface Homme Machine, le "trois tiers" pour des applications de type systèmes d'information web, le "BCED" [Mac01] pour des applications centrées données (patron dérivé du "Derived Entity Control" [Evi00]).

La figure B.2 représente le patron d'architecture "BCED" et sa décomposition en 4 couches : "Boundary" pour la présentation ; "Control" pour la logique de l'application ; "Entity" (fonctionnalités métiers) pour gérer l'intégrité logique ; la dernière étant "Data" pour l'accès et le stockage des données (Base de données). L'accès aux données sépare la Base de Données des données métier manipulées par le cœur de l'application (Entity).

Un dernier exemple est le "Presentation Abstraction Control". Sa parentalité revient à Joelle Coutaz[Cou87]. La structure du patron PAC propose une coopération hiérarchique de plusieurs

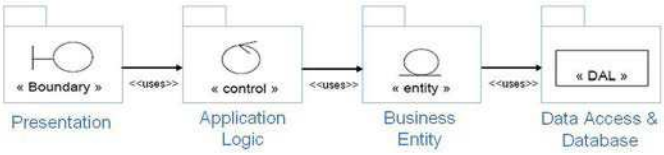


FIGURE B.2: Vue d'ensemble du patron BCED.

agents (dit "*top-level*", "*intermediate-level*" et "*bottom-level*"), chaque agent ayant une composante de "*presentation*" (description de son comportement visible), une composante "*abstraction*" (gestion du modèle de données et des fonctionalités agissant sur celles-ci) et "*control*" (assurant une connection entre la partie présentation et abstraction, ainsi que la communication avec les autres agents).

Les styles architecturaux

Perry *et al.* introduisent dans [PW92] la notion de "*architectural style*" - de style architectural - tel que :

> *"An architectural style defines a family of software systems in terms of their structural organisation. An architectural style expresses components and the relationships between them with the constraints of their application, and the associated composition and design rules for their construction."*
>
> Perry *et al.* [PW92](traduction p. 206)

Un style architectural exprime un type particulier de structure fondamentale (patron typologique) combiné avec une méthode spécifiant comment le construire (propriétés sémantiques). La spécification de la structure repose sur des contraintes et un vocable cohérent de désignation des éléments. Les styles ne sont pas des architectures complètes, mais davantage des protoarchitectures. Un style architectural est très proche d'un patron d'architecture, en effet, tout style architectural peut être décrit par un patron. Les patrons sont plus orientés problème que les style architecturaux. Certains évoqueront le fait que la généricité des styles est plus grande, car se basant sur un point de vue indépendant d'une situation de réalisation effective. De nombreux styles ont été définis [AAG95, SC97].

B.3.2 Évolution d'une *ligne de produits*

La taxonomie de l'évolution des *lignes de produits* de Svahnberg et Bosch [SB99] est représentée figure B.3.

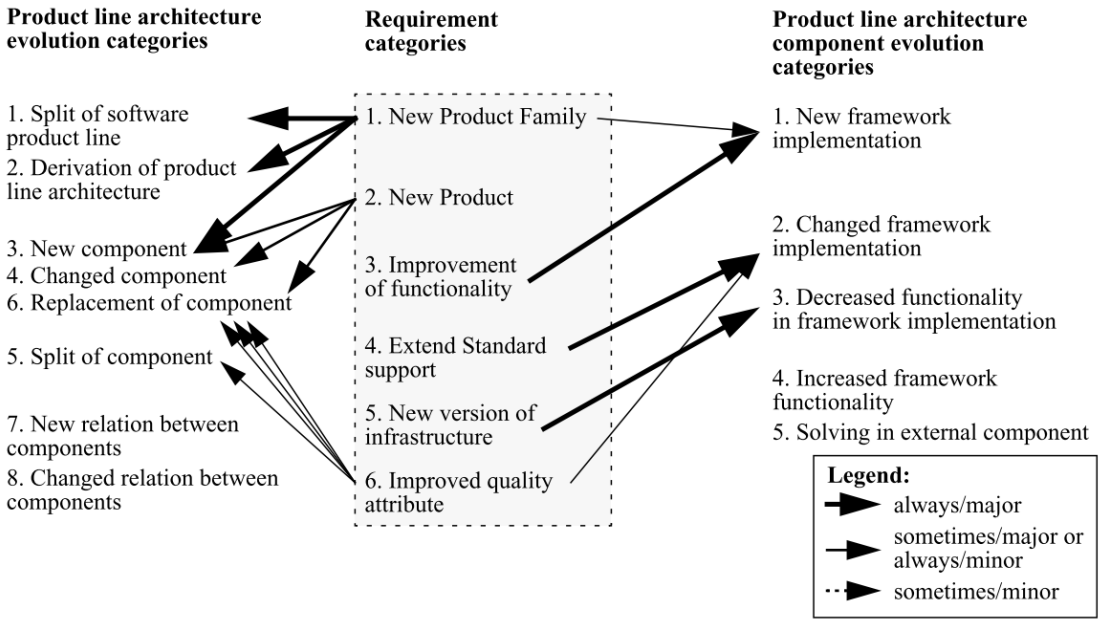


Figure B.3: Relations entre les catégories d'évolutions[SB99].

ANNEXE C

Compléments à la validation et à l'outillage

Bibliographie et Sitographie

Bibliographie

[AAG95] Gregory D. Abowd, Robert Allen, and David Garlan. Formalizing style to understand descriptions of software architecture. *ACM Trans. Softw. Eng. Methodol.*, 4(4):319–364, October 1995. *Cité page 217*

[ABB+02] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst, and Jörg Zettel. *Component-based product line engineering with UML*. Addison-Wesley Longman Pub. Co., Inc., Boston, US, 2002. *4 citations pages 40, 65, 67, et 139*

[AC04] Michal Antkiewicz and Krzysztof Czarnecki. Featureplugin: feature modeling plug-in for eclipse. In Michael G. Burke, editor, *ETX*, pages 67–72. ACM, 2004. *Cité page 45*

[ACC+02] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, October 2002. *Cité page 31*

[Ach11] Mathieu Acher. *Managing Multiple Feature Models: Foundations, Language, and Applications*. PhD thesis, Université de Nice Sophia Antipolis, 2011. *2 citations pages 49 et 72*

[ACLF10] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. Composing feature models. In *Proceedings of the SLE Conference*, volume 5969 of *LNCS*, pages 62–81. Springer, 2010. *2 citations pages 4 et 49*

[AG01] Michalis Anastasopoulos and Critina Gacek. Implementing product line variabilities. In *Proceedings of the 2001 symposium on Software reusability: putting software reuse in context*, SSR '01, pages 109–117, New York, NY, USA, 2001. ACM. *Cité page 41*

[AG10] L. Ackerman and C. Gonzalez. *Patterns-Based Engineering: Successfully Delivering Solutions Via Patterns*. Addison-Wesley, 2010. *2 citations pages 66 et 215*

[AGM+06] Vander Alves, Rohit Gheyi, Tiago Massoni, Uirá Kulesza, Paulo Borba, and Carlos Lucena. Refactoring product lines. In *Proceedings of the 5th international conference on Generative programming and component engineering*, GPCE '06, pages 201–210, New York, NY, USA, 2006. ACM. *Cité page 5*

[AJTK09] Sven Apel, Florian Janda, Salvador Trujillo, and Christian Kästner. Model superimposition in software product lines. In *Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations*, ICMT '09, pages 4–19, Berlin, Heidelberg, 2009. Springer-Verlag. *2 citations pages 40 et 41*

[AK01] Colin Atkinson and Thomas Kühne. The essence of multilevel metamodeling. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, UML'01, pages 19–33, London, UK, UK, 2001. Springer-Verlag. *Cité page 23*

[AK02] Colin Atkinson and Thomas Kühne. Rearchitecting the uml infrastructure. *ACM Trans. Model. Comput. Simul.*, 12(4):290–321, October 2002. *Cité page 23*

[AK08] Samuel A. Ajila and Ali B. Kaba. Evolution support mechanisms for software product line process. *J. Syst. Softw.*, 81(10):1784–1801, October 2008. *2 citations pages 55 et 74*

[AKM+10] Nicolas Anquetil, Uirá Kulesza, Ralf Mitschke, Ana Moreira, Jean-Claude Royer, Andreas Rummler, and André Sousa. A model-driven traceability framework for software product lines. *SoSyM*, 9:427–451, 2010. *3 citations pages 32, 55, et 177*

[Ale79] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979. *Cité page 215*

[Ale02] Ian Alexander. Towards automatic traceability in industrial practice. In *In Proc. of the 1st Int. Workshop on Traceability*, pages 26–31, 2002. *Cité page 29*

[ALMK08] Sven Apel, Christian Lengauer, Bernhard Möller, and Christian Kästner. An algebra for features and feature composition. In *Proceedings of the 12th international conference on Algebraic Methodology and Software Technology*, AMAST 2008, pages 36–50, Berlin, Heidelberg, 2008. Springer-Verlag. *2 citations pages 45 et 206*

[AMR02] P. Arkley, P. Mason, and S. Riddle. Position paper: Enabling traceability. In *Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 61–65, Edinburgh, Scotland, UK, September 2002. *Cité page 29*

[AP03] Marcus Alanen and Ivan Porres. Difference and union of models. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *"UML" 2003 - The Unified Modeling Language. Modeling Languages and Applications*, volume 2863 of *Lecture Notes in Computer Science*, pages 2–17. Springer Berlin / Heidelberg, 2003. *Cité page 161*

[AR05] Paul Arkley and Steve Riddle. Overcoming the traceability benefit problem. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, RE '05, pages 385–389, Washington, DC, USA, 2005. IEEE Computer Society. *Cité page 32*

[ARNRSG06a] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Syst. J.*, 45(3):515–526, July 2006. *2 citations pages 32 et 72*

[ARNRSG06b] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Syst. J.*, 45(3):515–526, July 2006. *Cité page 30*

[ASM04] Timo Asikainen, Timo Soininen, and Tomi Männistö. *A Koala-Based Approach for Modelling and Deploying Configurable Software Product Families*, volume 3014 of *Lecture Notes in Computer Science*, pages 225–249. Springer, 2004. *6 citations pages 38, 39, 54, 57, 65, et 211*

[BA96] Shawn A. Bohner and Robert S. Arnold. *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996. *Cité page 74*

[Bab86] Wayne A. Babich. *Software configuration management: coordination for team productivity*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986. *Cité page 72*

[Bai93] S. Bailin. Domain analysis with kaptur. *Tutorials of TRI-Ada'93*, 1993. *2 citations pages 44 et 49*

[BAS08] Don Batory, Maider Azanza, and João Saraiva. The objects and arrows of computational design. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, MoDELS '08, pages 1–20, Berlin, Heidelberg, 2008. Springer-Verlag. *Cité page 62*

[Bat05a] Don S. Batory. Feature models, grammars, and propositional formulas. In J. Henk Obbink and Klaus Pohl, editors, *SPLC*, volume 3714 of *LNCS*, pages 7–20. Springer, 2005. *3 citations pages 45, 46, et 205*

[Bat05b] Don S. Batory. Feature oriented programming for product-lines. Presented at: Summer School on Generative and Transformational Techniques in Software Engineering, July 4-8, Braga, Portugal, July 2005. *2 citations pages 45 et 205*

[BBG+90] D. S. Batory, J. R. Barnett, J. F. Garza, K. P. Smith, K. Tsukuda, B. C. Twichell, and T. E. Wise. Readings in object-oriented database systems. In Stanley B. Zdonik and David Maier, editors, *Readings in object-oriented database systems*, chapter GENESIS: an extensible database management system, pages 500–518. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. *Cité page 44*

[BBH+96] C. Boldyreff, E. L. Burd, R. M. Hather, M. Mum-o, and E. J. Younger. Greater understanding through maintainer driven traceability. In *Proceedings of the 4th International Workshop on Program Comprehension (WPC '96)*, WPC '96, pages 100–, Washington, DC, USA, 1996. IEEE Computer Society. *Cité page 30*

[BBM05] Kathrin Berg, Judith Bishop, and Dirk Muthig. Tracing software product line variability: from problem to solution space. In *Proceedings of the SAICSIT '05*, pages 182–191, 2005. *Cité page 55*

[BC00] C.Y. Baldwin and K.B. Clark. *Design Rules: The Power of Modularity*. Design Rules. Mit Press, 2000. *Cité page 190*

[BC05] Felix Bachmann and Paul Clements. Variability in software product lines. Technical report, Software Engineering Institute, Carnegie Mellon University, 2005. Technical Report CMU/SEI-2005-TR-012. *2 citations pages 37 et 205*

[BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edition, 2003. *4 citations pages 62, 67, 143, et 214*

[BCTS06] David Benavides, Antonio Ruiz Cortés, Pablo Trinidad, and Sergio Segura. A survey on the automated analyses of feature models. In José Cristóbal Riquelme Santos and Pere Botella, editors, *XI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2006), Octubre 3-6, 2006, Sitges, Barcelona, Spain*, pages 367–376, 2006. *Cité page 134*

[BCW11] Kacper Bak, Krzysztof Czarnecki, and Andrzej Wasowski. Feature and meta-models in clafer: mixed, specialized, and coupled. In *Proceedings of the Third international conference on Software language engineering*, SLE'10, pages 102–122, Berlin, Heidelberg, 2011. Springer-Verlag. *Cité page 40*

[BDFB07] Mikaël Barbero, Marcos Didonet, Del Fabro, and Jean Bézivin. Traceability and provenance issues in global model management. In *Proceedings of the 3rd ECMDA-Traceability Workshop*, 2007. *3 citations pages 24, 25, et 32*

[Ben02] PerOlof Bengtsson. *Architecture-Level Modifiability Analysis*. Ph.d. thesis, Blekinge Institute of Technology, Sweden, 2002. Dissertation series No 2002-2. *Cité page 140*

[Beu03] Danilo Beuche. Variant management with pure::variant. Technical report, pure-systems GmbH, 2003. *3 citations pages 41, 62, et 68*

[Beu08] Danilo Beuche. Modeling and building software product lines with pure::variants. *Software Product Line Conference, International*, 0:358, 2008. *3 citations pages 41, 45, et 68*

[BFG+02] Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, J. Henk Obbink, and Klaus Pohl. Variability issues in software product lines. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 13–21, London, UK, UK, 2002. Springer-Verlag. *Cité page 3*

[BFK+99] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc DeBaud. Pulse: a methodology to develop software product lines. In *Proceedings of the 1999 symposium on Software reusability*, SSR '99, pages 122–131, New York, NY, USA, 1999. ACM. *2 citations pages 65 et 277*

[BFW01] F. Boniol, J. Foisseau, and V. Wiels. Un exemple de modèle conceptuel de référence pour l'ingénierie des systèmes avioniques. In *2ème Conférence annuelle d'Ingénierie Système*, 2001. *Cité page 21*

[BG01] Jean Bézivin and Olivier Gerbé. Towards a precise definition of the omg/mda framework. In *Proceedings of the 16th IEEE international conference on Automated software engineering*, ASE '01, pages 273–, Washington, DC, USA, 2001. IEEE Computer Society. *2 citations pages 16 et 203*

[Big89] Ted J. Biggerstaff. Design recovery for maintenance and reuse. *Computer*, 22(7):36–49, July 1989. *Cité page 44*

[BJV04] Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. On the need for megamodels. In *Proceedings of the OOPSLA/GPCE : Best Practices for Model-Driven Software Developent workshop*, 2004. *3 citations pages 24, 25, et 204*

[BK05] Jean Bézivin and Ivan Kurtev. Model-based technology integration with the technical space concept. In *In: Proceedings of the Metainformatics Symposium, Springer-Verlag*. Springer-Verlag, 2005. *Cité page 23*

[BL09] Goetz Botterweck and Kwanwoo Lee. Feature dependencies have to be managed throughout the whole product life-cycle. In Jürgen Münch and Peter Liggesmeyer, editors, *Software Engineering 2009 - Workshopband, Fachtagung des*

GI-Fachbereichs Softwaretechnik 02.-06.03.2009 in Kaiserslautern, LNI, pages 101–106. GI, 2009. *Cité page 59*

[BLHM02] Don Batory, Roberto E. Lopez-Herrejon, and Jean-Philippe Martin. Generating product-lines of product-families. In *Proceedings of the 17th IEEE international conference on Automated software engineering*, ASE '02, pages 81–, Washington, DC, USA, 2002. IEEE Computer Society. *Cité page 41*

[BLP05] S. Buhne, K. Lauenroth, and K. Pohl. Modelling requirements variability across product lines. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, pages 41 – 50, aug.-2 sept. 2005. *Cité page 41*

[BMB+11] Marko Bošković, Gunter Mussbacher, Ebrahim Bagheri, Daniel Amyot, Dragan Gašević, and Marek Hatala. Aspect-oriented feature models. In Juergen Dingel and Arnor Solberg, editors, *Models in Software Engineering*, volume 6627 of *Lecture Notes in Computer Science*, pages 110–124. Springer Berlin Heidelberg, 2011. *3 citations pages 53, 57, et 210*

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Number vol. 1 in Pattern-Oriented Software Architecture. Wiley, volume 1 edition, 1996. *5 citations pages 141, 142, 145, 215, et 216*

[Boh02] Shawn A. Bohner. Extending software change impact analysis into cots components. In *Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02)*, SEW '02, pages 175–, Washington, DC, USA, 2002. IEEE Computer Society. *Cité page 74*

[Bos99] Jan Bosch. Evolution and composition of reusable assets in product-line architectures: A case study. In *Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, WICSA1, pages 321–340, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V. *Cité page 74*

[Bos00] Jan Bosch. *Design and use of software architectures: adopting and evolving a product-line approach*. ACM Press/Addison-Wesley Pub. Co., NY, USA, 2000. *10 citations pages 5, 45, 61, 63, 64, 65, 67, 72, 145, et 205*

[Bos02] Jan Bosch. Maturity and evolution in software product lines: Approaches, artefacts and organization. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 257–271, London, UK, UK, 2002. Springer-Verlag. *2 citations pages 67 et 72*

[Bos09] Jan Bosch. From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 111–119, Pittsburgh, PA, USA, 2009. Carnegie Mellon University. *Cité page 79*

[BPD+10] Goetz Botterweck, Andreas Pleuss, Deepak Dhungana, Andreas Polzer, and Stefan Kowalewski. Evofm: feature-driven planning of product-line evolution. In *Proceedings of the 2010 ICSE Workshop on Product Line Approaches in Software Engineering*, PLEASE '10, pages 24–31, New York, NY, USA, 2010. ACM. *2 citations pages 72 et 74*

[BR08] Robert Brcina and Matthias Riebisch. Defining a traceability link semantics for design decision support. In Jon Oldevik, Gøran K. Olsen, Tor Neple, and Richard Paige, editors, *ECMDA Traceability Workshop Proceedings 2008*, pages 39 – 48. SINTEF ICT, 2008. ISBN 978-82-14-04396. *Cité page 32*

[BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language user guide*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1999. *2 citations pages 16 et 203*

[Bro04] A. W. Brown. Model driven architecture: principles and practice. *SoSyM*, 3(3):314–327, 2004. *2 citations pages 16 et 203*

[BSC10] David Benavides, Sergio Segura, and Antonio Ruiz Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, 2010. *Cité page 46*

[BSP09] Goetz Botterweck, Denny Schneeweiss, and Andreas Pleuss. Interactive techniques to support the configuration of complex feature models. In *MDPLE 2009*, 2009. *Cité page 4*

[BSR03] Don Batory, Jacob Neal Sarvela, and Axel Rauschmayer. Scaling step-wise refinement. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 187–197, Washington, DC, USA, 2003. IEEE Computer Society. *2 citations pages 41 et 84*

[BTN+08] Goetz Botterweck, Steffen Thiel, Daren Nestor, Saad bin Abid, and Ciarán Cawley. Visual tool support for configuring and understanding software product lines. In *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC '08, pages 77–86, Washington, DC, USA, 2008. IEEE Computer Society. *5 citations pages 48, 49, 52, 57, et 210*

[BW02] Joachim Bayer and Tanya Widen. Introducing traceability to product lines. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 409–416, London, UK, UK, 2002. Springer-Verlag. *2 citations pages 30 et 55*

[CA05] Krzysztof Czarnecki and Michal Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In Robert Glück and Michael R. Lowry, editors, *Generative Programming and Component Engineering, 4th International Conference, GPCE 2005, Tallinn, Estonia, September 29 - October 1, 2005, Proceedings*, volume 3676 of *Lecture Notes in Computer Science*, pages 422–437. Springer, 2005. *7 citations pages 4, 40, 41, 53, 57, 68, et 210*

[CABA09] Lianping Chen, Muhammad Ali Babar, and Nour Ali. Variability management in software product lines: a systematic review. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 81–90, Pittsburgh, PA, USA, 2009. Carnegie Mellon University. *Cité page 38*

[CAK+05] Krzysztof Czarnecki, Michal Antkiewicz, Chang Hwan Peter Kim, Sean Lau, and Krzysztof Pietroszek. Model-driven software product lines. In *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '05, pages 126–127, New York, NY, USA, 2005. ACM. *2 citations pages 3 et 62*

[CAS03] Gilberto A. A. Cysneiros, Filho Andrea, and Zisman George Spanoudakis. Traceability approach for i* and uml models. In *in Proceedings of 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'03*, 2003. *Cité page 29*

[CBH11] Andreas Classen, Quentin Boucher, and Patrick Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Science of Computer Programming*, 76(12):1130 – 1143, 2011. *4 citations pages 54, 57, 59, et 211*

[CC12] Stephen Creff and Joël Champeau. Relationships in variability modeling approaches: A survey and classification. In *Proceedings of Journée Lignes de Produits '12*, Lille, France, 2012. *2 citations pages 5 et 50*

[CC13] Stephen Creff and Joël Champeau. A dsml to reify the relations among artifacts in model-based product lines. In *Proceedings of the ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems - MoDELS volume II, First International Workshop SIMF: Semantic Information Modeling for Federation*, Miami, Florida - USA, 2013. to appear. *3 citations pages 5, 107, et 127*

[CCJM12a] Stephen Creff, Joël Champeau, Jean-Marc Jézéquel, and Arnaud Monégier. Model-based product line evolution: an incremental growing by extension. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*, SPLC '12, pages 107–114, New York, NY, USA, 2012. ACM. *2 citations pages 6 et 151*

[CCJM12b] Stephen Creff, Joël Champeau, Jean-Marc Jézéquel, and Arnaud Monégier. Relationships formalization for model-based product lines. In *Proceedings of APSEC '12*, Hong Kong, 2012. to appear. *5 citations pages 5, 83, 95, 107, et 127*

[CCMJ12] Stephen Creff, Joël Champeau, Arnaud Monégier, and Jean-Marc Jézéquel. Une organisation des lignes de produits logiciels autour d'un motif architectural. In *Actes de CAL '12*, Montpellier, France, 2012. *3 citations pages 6, 140, et 169*

[CdCMdM+11] Yguaratã Cerqueira Cavalcanti, Ivan do Carmo Machado, Paulo Anselmo da Mota, Silveira Neto, Luanna Lopes Lobato, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. Towards metamodel support for variability and traceability in software product lines. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 49–57, New York, NY, USA, 2011. ACM. *Cité page 55*

[CE00] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Pub. Co., NY, NY, USA, 2000. *8 citations pages 40, 41, 44, 45, 51, 57, 205, et 209*

[CESW04] Tony Clark, Andy Evans, Paul Sammut, and James Willans. *Applied Metamodelling - A Foundation for Language Driven Development*. unknown, second edition, 2004. *3 citations pages 2, 13, et 21*

[CGR+12] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. Cool features and tough decisions: a comparison of variability modeling approaches. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 173–182, New York, NY, USA, 2012. ACM. *Cité page 39*

[CH06] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45(3):621–645, July 2006. *Cité page 28*

[CHBT10] Ciarán Cawley, Patrick Healy, Goetz Botterweck, and Steffen Thiel. Research tool to support feature configuration in software product lines. In Paul Grünbacher David Benavides, Don Batory, editor, *Proceedings of 4th International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS 2010)*, pages 179–182, 2010. *Cité page 45*

[CHCS+02] Jane Cleland-Huang, Carl K. Chang, Gaurav Sethi, Kumar Javvaji, Haijian Hu, and Jinchun Xia. Automating speculative queries through event-based requirements traceability. In *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, RE '02, pages 289–298, Washington, DC, USA, 2002. IEEE Computer Society. *Cité page 31*

[CHE04] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration using feature models. In Robert L. Nord, editor, *Software Product Lines*, volume 3154 of *Lecture Notes in Computer Science*, pages 162–164. Springer Berlin / Heidelberg, 2004. *3 citations pages 4, 65, et 68*

[CHE05] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005. *Cité page 45*

[CHGZ12] Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman, editors. *Software and Systems Traceability*. Springer London, London, 2012. *Cité page 31*

[CHS08] Andreas Classen, Patrick Heymans, and Pierre-Yves Schobbens. What's in a feature: a requirements engineering perspective. In *Proceedings of the Theory and practice of software, 11th international conference on Fundamental approaches to software engineering*, FASE'08/ETAPS'08, pages 16–30, Berlin, Heidelberg, 2008. Springer-Verlag. *Cité page 38*

[CHS+10] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, and Jean-François Raskin. Model checking lots of systems: efficient verification of temporal properties in software product lines. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 335–344, New York, NY, USA, 2010. ACM. *4 citations pages 53, 57, 59, et 210*

[CHW98] James Coplien, Daniel Hoffman, and David Weiss. Commonality and variability in software engineering. *IEEE Softw.*, 15(6):37–45, November 1998. *Cité page 65*

[CK02] Paul Clements and Charles Kreuger. Point-counterpoint. *IEEE Software*, 19:28–31, 2002. *Cité page 72*

[CKI88] Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Commun. ACM*, 31(11):1268–1287, November 1988. *Cité page 199*

[CKM02] Kester Clegg, Tim Kelly, and John McDermid. Incremental product-line development. In *International Workshop on PLE*, Seattle, 2002. *Cité page 74*

[Cla01] M. Clauss. Generic modeling using uml extensions for variability. In *In Proceedings of OOPSLA Workshop on Domain-specific Visual Languages*, pages 11–18, Tampa, FL, USA, 2001. *Cité page 40*

[Cla11] Mickaël Clavreul. *Composition de modèles et de métamodèles : Séparation des correspondances et des interpretations pour unifier les approches de composition existantes*. PhD thesis, Université de Rennes 1, 2011. *Cité page 28*

[Cle01] Paul C. Clements. Component-based software engineering. In George T. Heineman and William T. Councill, editors, *Component-based software engineering*, chapter From subroutines to subsystems: component-based software development, pages 189–198. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. *Cité page 65*

[CLK08] Hojin Cho, Kwanwoo Lee, and Kyo Chul Kang. Feature relation and dependency management: An aspect-oriented approach. In *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC '08, pages 3–11, Washington, DC, USA, 2008. IEEE Computer Society. *3 citations pages 52, 57, et 210*

[CLLK09] Hyunsik Choi, Kwanwoo Lee, Jaejoon Lee, and Kyo C. Kang. Multiple views of feature models to manage complexity. In *Proceedings of the International Workshop on Scalable Modeling Techniques for Software Product Lines (SCALE 2009)*, pages 127–133, San Francisco, California, USA, 2009. *3 citations pages 53, 57, et 210*

[CM98] Charles Consel and Renaud Marlet. Architecture software using: A methodology for language development. In *Proceedings of the 10th International Symposium on Principles of Declarative Programming*, PLILP '98/ALP '98, pages 170–194, London, UK, UK, 1998. Springer-Verlag. *Cité page 18*

[CMCJ11] Stephen Creff, Arnaud Monégier, Joël Champeau, and Jean-Marc Jézéquel. Multiple feature models & relationships in a model-based product line process. In *4ème Journée Lignes de Produits*. Université Paris I Panthéon-Sorbonne, 2011. *3 citations pages 5, 83, et 169*

[CN01] Paul C. Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001. *12 citations pages 35, 61, 62, 63, 64, 65, 67, 72, 73, 139, 216, et 277*

[CNS11] Marsha Chechik, Shiva Nejati, and Mehrdad Sabetzadeh. A relationship-based approach to model integration. *Innovations in Systems and Software Engineering*, pages 1–16, 2011. *2 citations pages 5 et 28*

[Coc97] Alistair Cockburn. Structuring use cases with goals. *Journal of Object-Oriented Programming*, 1997. *Cité page 144*

[Con93] Software Productivity Consortium. *Reuse-driven Software Processes Guidebook*. Software Productivity Consortium, 1993. *Cité page 39*

[Cou87] Joëlle Coutaz. Pac, on object oriented model for dialog design. In *Interact'87*, 1987. 6 pages. *3 citations pages 142, 216, et 276*

[CP06] Krzysztof Czarnecki and Krzysztof Pietroszek. Verifying feature-based model templates against well-formedness ocl constraints. In Stan Jarzabek, Douglas C. Schmidt, and Todd L. Veldhuizen, editors, *Generative Programming and Component Engineering, 5th International Conference, GPCE 2006, Portland, Oregon, USA, October 22-26, 2006, Proceedings*, pages 211–220. ACM, 2006. *2 citations pages 41 et 43*

[CREP08] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. Automating co-evolution in model-driven engineering. In *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, EDOC '08, pages 222–231, Washington, DC, USA, 2008. IEEE Computer Society. *Cité page 71*

[CSW08] Krzysztof Czarnecki, Steven She, and Andrzej Wasowski. Sample spaces and feature models: There and back again. In *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC '08, pages 22–31, Washington, DC, USA, 2008. IEEE Computer Society. *4 citations pages 53, 57, 59, et 210*

[CTBN09] Ciarán Cawley, Steffen Thiel, Goetz Botterweck, and Daren Nestor. Visualising inter-model relationships in software product lines. In David Benavides, Andreas Metzger, and Ulrich W. Eisenecker, editors, *Third International Workshop on Variability Modelling of Software-Intensive Systems, Seville, Spain, January 28-30, 2009. Proceedings*, volume 29 of *ICB Research Report*, pages 37–44. Universität Duisburg-Essen, 2009. *3 citations pages 52, 57, et 210*

[CTH08] Ciarán Cawley, Steffen Thiel, and Patrick Healy. Visualising variability relationships in software product lines. In Steffen Thiel and Klaus Pohl, editors, *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings. Second Volume (Workshops)*, pages 329–333. Lero Int. Science Centre, University of Limerick, Ireland, 2008. *Cité page 49*

[CZZM05] Kun Chen, Wei Zhang, Haiyan Zhao, and Hong Mei. An approach to constructing feature models based on requirements clustering. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, RE '05, pages 31–40, Washington, DC, USA, 2005. IEEE Computer Society. *2 citations pages 45 et 205*

[Del03] Alain Delaunay. Abstraction. Encyclopædia Universalis, 2003. *Cité page 14*

[DGD06] Dragan Djuric, Dragan Gašević, and Vladan Devedzic. The tao of modeling spaces. *Journal of Object Technology*, 5:125–147, 2006. *2 citations pages 14 et 22*

[DGF05] Stéphane Ducasse, Tudor Gîrba, and Jean-Marie Favre. Modeling software evolution by treating history as a first class entity. *Electronic Notes in Theoretical Computer Science*, 127(3):75 – 86, 2005. Proceedings of the Workshop on Software Evolution through Transformations: Model-based vs. Implementation-level Solutions (SETra 2004). *Cité page 72*

[DGR11] Deepak Dhungana, Paul Grünbacher, and Rick Rabiser. The dopler meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engg.*, 18(1):77–114, March 2011. *5 citations pages 41, 50, 54, 57, et 211*

[DGRN10] Deepak Dhungana, Paul Grünbacher, Rick Rabiser, and Thomas Neumayer. Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software*, 83(7):1108–1122, July 2010. *2 citations pages 5 et 74*

[Dic02] Jeremy Dick. Rich traceability. In *IEEE Software*, page 2005, 2002. *2 citations pages 29 et 30*

[Dij82] Edsger W. Dijkstra. *Selected Writings on Computing: A Personal Perspective*, chapter EWD 447: On the role of scientific thought (1974), pages 60–66. Springer-Verlag, New York, New York, USA, 1982. *Cité page 18*

[dLG10] Juan de Lara and Esther Guerra. Deep meta-modelling with metadepth. In *Proceedings of the 48th international conference on Objects, models, components, patterns*, TOOLS'10, pages 1–20, Berlin, Heidelberg, 2010. Springer-Verlag. *Cité page 23*

[DLS05] Gan Deng, Gunther Lenz, and Douglas C. Schmidt. Addressing domain evolution challenges in software product lines. In J.-M. Bruel, editor, *MoDELS Satellite Events*, volume 3844 of LNCS, pages 247–261. Springer, 2005. *Cité page 74*

[DNGR08a] Deepak Dhungana, Thomas Neumayer, Paul Grunbacher, and Rick Rabiser. Supporting evolution in model-based product line engineering. In *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC '08, pages 319–328, Washington, DC, USA, 2008. IEEE Computer Society. *Cité page 74*

[DNGR08b] Deepak Dhungana, Thomas Neumayer, Paul Grünbacher, and Rick Rabiser. Supporting the evolution of product line architectures with variability model fragments. In *Proceedings of the WICSA 2008*, pages 327–330, Washington, DC, USA, 2008. IEEE Computer Society. *2 citations pages 74 et 75*

[dOGHM05] Edson Alves de Oliveira, Junior, Itana M. S. Gimenes, Elisa Hatsue Moriya Huzita, and José Carlos Maldonado. A variability management process for software product lines. In *Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '05, pages 225–241. IBM Press, 2005. *Cité page 40*

[DP05] Åsa Dahlstedt and Anne Persson. Requirements interdependencies: State of the art and future challenges. In Aybüke Aurum and Claes Wohlin, editors, *Engineering and Managing Software Requirements*, pages 95–116. Springer Berlin Heidelberg, 2005. *Cité page 29*

[DRM00] Ebru Dincel, Roshanak Roshandel, and Nenad Medvidovic. Adl independent architectural representation in xml, 2000. *Cité page 67*

[DS09] Luo Daizhong and Diao Shanhui. Feature dependency modeling for software product line. In *Computer Engineering and Technology, 2009. ICCET '09. International Conference on*, volume 2, pages 256 –260, jan. 2009. *4 citations pages 52, 57, 59, et 210*

[DSB04] Sybren Deelstra, Marco Sinnema, and Jan Bosch. Experiences in software product families: Problems and issues during product derivation. In Robert Nord, editor, *Software Product Lines*, volume 3154 of *Lecture Notes*

in Computer Science, pages 120–122. Springer Berlin / Heidelberg, 2004. *2 citations pages 68 et 69*

[DSB05] Sybren Deelstra, Marco Sinnema, and Jan Bosch. Product derivation in software product families: a case study. *J. Syst. Softw.*, 74:173–194, January 2005. *4 citations pages 5, 57, 69, et 76*

[DSB09] Sybren Deelstra, Marco Sinnema, and Jan Bosch. Variability assessment in software product families. *Inf. Softw. Technol.*, 51(1):195–218, January 2009. *5 citations pages 38, 54, 57, 74, et 211*

[DSB+11] Deepak Dhungana, Dominik Seichter, Goetz Botterweck, Rick Rabiser, Paul Grunbacher, David Benavides, and Jose A. Galindo. Configuration of multi product lines by bridging heterogeneous variability modeling approaches. In *Proceedings of the 2011 15th International Software Product Line Conference*, SPLC '11, pages 120–129, Washington, DC, USA, 2011. IEEE Computer Society. *3 citations pages 55, 57, et 211*

[EBB05] Magnus Eriksson, Jürgen Börstler, and Kjell Borg. The pluss approach - domain modeling with features, use cases and use case realizations. In Henk Obbink and Klaus Pohl, editors, *Software Product Lines*, volume 3714 of *Lecture Notes in Computer Science*, pages 33–44. Springer Berlin / Heidelberg, 2005. *3 citations pages 40, 65, et 277*

[EBLSP10] Christoph Elsner, Goetz Botterweck, Daniel Lohmann, and Wolfgang Schröder-Preikschat. Variability in time - product line variability and evolution revisited. In *VaMoS'10*, pages 131–137, 2010. *2 citations pages 37 et 72*

[EDL10] Jacky Estublier, Idrissa Dieng, and Thomas Lévêque. Software product line evolution: The selecta system. In *PLEASE '10: Proceedings of the 2010 ICSE Workshop on Product Line Approaches in Software Engineering*, pages 32–39, New York, NY, USA, May 2010. ACM. *Cité page 74*

[EG02] Alexander Egyed and Paul Grünbacher. Automating requirements traceability: Beyond the record & replay paradigm. In *Proceedings of the 17th IEEE international conference on Automated software engineering*, ASE '02, pages 163–, Washington, DC, USA, 2002. IEEE Computer Society. *Cité page 29*

[Egy03] Alexander Egyed. A scenario-driven approach to trace dependency analysis. *IEEE Trans. Softw. Eng.*, 29(2):116–132, February 2003. *2 citations pages 30 et 31*

[Elk01] W.-El Elkaim. System family architecture glossary. Technical report, ESAPS Project, 2001. *Cité page 64*

[Eng87] Y. Engeström. *Learning by Expanding: An Activity-theoretical Approach to Developmental Research*. Orienta-Konsultit Oy, 1987. *Cité page 33*

[ES08] Leire Etxeberria and Goiuria Sagardui. Variability driven quality evaluation in software product lines. In *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC '08, pages 243–252, Washington, DC, USA, 2008. IEEE Computer Society. *Cité page 38*

[Etz64] Amitai Etzioni. *Modern organizations*. Foundations of modern sociology series. Prentice-Hall, Englewood Cliffs, N.J., 1964. *Cité page 215*

[Evi00] Paul Evitts. *A UML Pattern Language*. New Riders Publishing, Thousand Oaks, CA, USA, 2000. *2 citations pages 216 et 276*

[Fau01] Stuart R. Faulk. Product-line requirements specification (prs): An approach and case study. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, RE '01, pages 48–, Washington, DC, USA, 2001. IEEE Computer Society. *Cité page 62*

[Fav05a] Jean-Marie Favre. Foundations of meta-pyramids: Languages vs. metamodels – episode ii: Story of thotus the baboon1. In Jean Bezivin and Reiko Heckel, editors, *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. *4 citations pages 2, 22, 25, et 91*

[Fav05b] Jean-Marie Favre. Foundations of model (driven) (reverse) engineering : Models – episode i: Stories of the fidus papyrus and of the solarus. In Jean Bezivin and Reiko Heckel, editors, *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. *7 citations pages 14, 16, 22, 24, 25, 26, et 204*

[Fav06] Jean-Marie Favre. Megamodelling and etymology. In James R. Cordy, Ralf Lämmel, and Andreas Winter, editors, *Transformation Techniques in Software Engineering*, number 05161 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. *2 citations pages 22 et 25*

[FB03] Karl Fogel and Moshe Bar. *Open Source Development with CVS*. Paraglyph Press, Scottsdale, AZ, 3. edition, 2003. *Cité page 72*

[FBFG07] Franck Fleurey, Benoit Baudry, Robert France, and Sudipto Ghosh. A generic approach for automatic model composition. In *Aspect Oriented Modeling (AOM) Workshop*, 2007. *4 citations pages 28, 69, 160, et 188*

[FEBF06] Jean-Marie Favre, Jacky Estublier, and Mireille Blay-Fornarino. *L'ingénierie dirigée par les modèles: au-delà du MDA*. Traité IC2 - Informatique et systèmes d'information. Hermès Science Publications, 2006. *3 citations pages 24, 25, et 271*

[Fei07] P.H. Feiler. Modeling of system families. Technical note, Carnegie Mellon University, Software Engineering Institute, 2007. CMU/SEI-2007-TN-047. *Cité page 67*

[FFB02] Dániel Fey, Róbert Fajta, and András Boros. Feature modeling: A meta-model to enhance usability and usefulness. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 198–216, London, UK, UK, 2002. Springer-Verlag. *3 citations pages 51, 57, et 210*

[FGH+94] A.C.W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *Software Engineering, IEEE Transactions on*, 20(8):569 –578, aug 1994. *Cité page 21*

[FHMP+09] Franck Fleurey, Østein Haugen, Birger Møller-Pedersen, Gøran K. Olsen, Andreas Svendsen, and Xiaorui Zhang. A generic language and tool for variability modeling. Technical report, SINTEF, 2009. *3 citations pages 41, 69, et 276*

[FHMP+12] Franck Fleurey, Øystein Haugen, Birger Møller-Pedersen, Andreas Svendsen, and Xiaorui Zhang. Standardizing variability - challenges and solutions. In *SDL 2011 Proceedings*, volume 7083 of *LNCS*, pages 233–246. Springer / Heidelberg, 2012. *Cité page 57*

[FHS02] Stefan Ferber, Jürgen Haag, and Juha Savolainen. Feature interaction and dependencies: Modeling features for reengineering a legacy product line. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 235–256. Springer-Verlag, London, UK, UK, 2002. *3 citations pages 51, 57, et 210*

[FLV12] Jean-Marie Favre, Ralf Lämmel, and Andrei Varanovich. Modeling the linguistic architecture of software products. In *Proceedings of MODELS 2012*, LNCS. Springer, 2012. 17 pages. To appear. *Cité page 25*

[FPDF98] William Frakes, Ruben Prieto- ;Diaz, and Christopher Fox. Dare: Domain analysis and reuse environment. *Annals of Software Engineering*, 5:125–141, 1998. *2 citations pages 65 et 276*

[FR07] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society. *2 citations pages 2 et 13*

[FV03] D. Faust and C. Verhoef. Software product line migration and deployment. *Software Practice and Experience, John Wiley & Sons, Ltd*, 33:933–955, 2003. *Cité page 74*

[GAM10] Yaser Ghanam, Darren Andreychuk, and Frank Maurer. Reactive variability management in agile software development. *AGILE Conference*, 0:27–34, 2010. *Cité page 74*

[GBLM04] Bruno Gonzalez-Baixauli, Julio Cesar Sampaio do Prado Leite, and John Mylopoulos. Visual variability analysis for goal models. In *Proceedings of the Requirements Engineering Conference, 12th IEEE International*, RE '04, pages 198–207, Washington, DC, USA, 2004. IEEE Computer Society. *Cité page 50*

[GBS01] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, WICSA '01, pages 45–, Washington, DC, USA, 2001. IEEE Computer Society. *2 citations pages 36 et 40*

[GCHH+12] Orlena Gotel, Jane Cleland-Huang, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, Alex Dekhtyar, Giulio Antoniol, and Jonathan Maletic. *The Grand Challenge of Traceability*, pages 343–412. Springer-Verlag London Limited, 2012. *Cité page 32*

[GEEM03] Paul Grünbacher, Alexander Egyed, Er Egyed, and Nenad Medvidovic. Reconciling software requirements and architectures with intermediate models. In *Software and Systems Modeling*, pages 202–211. Springer, 2003. *Cité page 143*

[GF94] O.C.Z. Gotel and C.W. Finkelstein. An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94 –101, apr 1994. *2 citations pages 30 et 32*

[GF95] O. Gotel and A. Finkelstein. Contribution structures [requirements artifacts]. *Requirements Engineering, IEEE International Conference on*, 0:100, 1995. *3 citations pages 29, 30, et 31*

[GFA98] M. L. Griss, J. Favaro, and M. d' Alessandro. Integrating feature modeling with the rseb. In *Proceedings of the 5th ICSR*. IEEE Computer Society, 1998. *10 citations pages 40, 44, 49, 50, 57, 65, 67, 139, 209, et 276*

[GG07] Ismênia Galvão and Arda Goknil. Survey of traceability approaches in model-driven engineering. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, EDOC '07, pages 313–, Washington, DC, USA, 2007. IEEE Computer Society. *Cité page 31*

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Co., Inc., Boston, MA, USA, 1995. *2 citations pages 215 et 216*

[GJM03] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, 2003. *Cité page 18*

[GKH07] Dragan Gašević, Nima Kaviani, and Marek Hatala. On metamodeling in megamodels. In Gregor Engels, Bill Opdyke, Douglas C. Schmidt, and Frank Weil, editors, *Model Driven Engineering Languages and Systems, 10th International Conference, MoDELS 2007, Nashville, USA, September 30 - October 5, 2007, Proceedings*, volume 4735 of *Lecture Notes in Computer Science*, pages 91–105. Springer, 2007. *2 citations pages 25 et 26*

[GKP98] Robert Geisler, Marcus Klar, and Claudia Pons. Dimensions and dichotomy in metamodeling. In *Proceedings of the 3rd BCS-FACS conference on Northern Formal Methods*, 3FACS'98, pages 10–10, Swinton, UK, UK, 1998. British Computer Society. *Cité page 23*

[GLR+02] Anna Gerber, Michael Lawley, Kerry Raymond, Jim Steel, and Andrew Wood. Transformation: The missing link of mda. In *Proceedings of the First International Conference on Graph Transformation*, ICGT '02, pages 90–105, London, UK, UK, 2002. Springer-Verlag. *Cité page 28*

[GLS08] Alexander Gruler, Martin Leucker, and Kathrin Scheidemann. Modeling and model checking software product lines. In *Proceedings of the 10th IFIP WG 6.1 international conference on Formal Methods for Open Object-Based Distributed Systems*, FMOODS '08, pages 113–131, Berlin, Heidelberg, 2008. Springer-Verlag. *Cité page 65*

[GMW97] David Garlan, Robert Monroe, and David Wile. Acme: an architecture description interchange language. In *Proceedings of the 1997 conference of the Centre*

for Advanced Studies on Collaborative research, CASCON '97, pages 7–. IBM Press, 1997. *Cité page 67*

[Gom04] Hassan Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004. *6 citations pages 40, 65, 67, 69, 141, et 276*

[GRDL09] Paul Grünbacher, Rick Rabiser, Deepak Dhungana, and Martin Lehofer. Structuring the product line modeling space: Strategies and examples. In David Benavides, Andreas Metzger, and Ulrich W. Eisenecker, editors, *Third International Workshop on Variability Modelling of Software-Intensive Systems, Seville, Spain, January 28-30, 2009. Proceedings*, volume 29 of *ICB Research Report*, pages 77–82. Universität Duisburg-Essen, 2009. *Cité page 49*

[Gri00] Martin L. Griss. Implementing product-line features with component reuse. In *Proceedings of the 6th International Conerence on Software Reuse: Advances in Software Reusability*, ICSR-6, pages 137–152, London, UK, UK, 2000. Springer-Verlag. *Cité page 76*

[GS02] Hassan Gomaa and Michael Eonsuk Shin. Multiple-view meta-modeling of software product lines. In *Proceedings of the Eighth International Conference on Engineering of Complex Computer Systems*, ICECCS '02, pages 238–, Washington, DC, USA, 2002. IEEE Computer Society. *Cité page 40*

[GSCK04] Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. John Wiley & Sons, 2004. *Cité page 23*

[GW92] Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artif. Intell.*, 57(2-3):323–389, October 1992. *2 citations pages 15 et 203*

[HC01] George T. Heineman and William T. Councill, editors. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. *2 citations pages 63 et 65*

[Hei09] Florian Heidenreich. Towards systematic ensuring well-formedness of software product lines. In *Proceedings of the First International Workshop on Feature-Oriented Software Development*, FOSD '09, pages 69–74, New York, NY, USA, 2009. ACM. *Cité page 53*

[HF08] Geir K. Hanssen and Tor E. Fígri. Process fusion: An industrial case study on agile software product line engineering. *J. Syst. Softw.*, 81(6):843–854, June 2008. *Cité page 74*

[HGKF03] Lothar Hotz, Andreas Günter, Thorsten Krebs, and Hitec C/o Fachbereich. A knowledge-based product derivation process and some ideas how to integrate product development. In *in of Software Variability Management Workshop*, pages 136–140, 2003. *Cité page 76*

[HHKN09] Soichiro Hidaka, Zhenjiang Hu, Hiroyuki Kato, and Keisuke Nakano. Towards a compositional approach to model transformation for software development. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 468–475, New York, NY, USA, 2009. ACM. *Cité page 28*

[HHU08] A. Hubaux, P. Heymans, and H. Unphon. Separating variability concerns in a product line re-engineering project. In *Proceedings of the 2008 AOSD workshop on Early aspects*, EA '08, pages 4:1–4:8, New York, NY, USA, 2008. ACM. *Cité page 53*

[HKW08] Florian Heidenreich, Jan Kopcsek, and Christian Wende. Featuremapper: Mapping features to models. In *Companion Proceedings of ICSE'08*. ACM, 2008. *7 citations pages 4, 40, 41, 53, 57, 68, et 210*

[HL95] Walter L. Hürsch and Cristina Videira Lopes. Separation of concerns. Technical report, College of Computer Science, Northeastern Univ., 1995. *Cité page 17*

[HLSS98] Peter van den Hamer, Frank van der Linden, Alison Saunders, and Henk Sligte. An integral hierarchy and diversity model for describing product family architecture. In *Proceedings of the Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families*, pages 66–75, London, UK, UK, 1998. Springer-Verlag. *Cité page 35*

[HMPO05] Øystein Haugen, Birger Møller-Pedersen, and Jon Oldevik. Comparison of system family modeling approaches. In Henk Obbink and Klaus Pohl, editors, *Software Product Lines*, volume 3714 of *Lecture Notes in Computer Science*, pages 102–112. Springer Berlin / Heidelberg, 2005. *Cité page 38*

[HMPO+08] Øystein Haugen, Birger Møller-Pedersen, Jon Oldevik, Gøran K. Olsen, and Andreas Svendsen. Adding standardized variability to domain specific languages. In *SPLC '08: Proceedings of the 2008 12th International Software Product Line Conference*, pages 139–148, Washington, DC, USA, 2008. IEEE Computer Society. *2 citations pages 40 et 41*

[HNS99] Christine Hofmeister, Robert L. Nord, and Dilip Soni. Describing software architecture with uml. In *Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, WICSA1, pages 145–160, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V. *Cité page 67*

[HP04] Günter Halmans and Klaus Pohl. Communicating the variability of a software-product family to customers. *Informatik - Forschung und Entwicklung*, 18:113–131, 2004. *Cité page 37*

[HR00] D. Harel and B. Rumpe. Modeling languages: Syntax, semantics and all that stuff, part i: The basic stuff. Technical report, Weizmann Institute of Science, Jerusalem, Israel, Israel, 2000. *Cité page 46*

[HR10] Wolfgang Heider and Rick Rabiser. Tool support for evolution of product lines through rapid feedback from application engineering. In David Benavides, Don S. Batory, and Paul Grünbacher, editors, *Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria, January 27-29, 2010. Proceedings*, volume 37 of *ICB-Research Report*, pages 167–170. Universität Duisburg-Essen, 2010. *Cité page 74*

[HRDG09] Wolfgang Heider, Rick Rabiser, Deepak Dhungana, and Paul Grünbacher. Tracking evolution in model-based product lines. In *Proceedings 1st Int'l Workshop on Model-driven Approaches in Software Product Line Engineering*

(MAPLE 2009), collocated with the 13th Int'l Software Product Line Conference (SPLC 2009), San Francisco, CA, USA, August 24, pages 59–63, 2009. *2 citations pages 55 et 74*

[HRR+11] Arne Haber, Holger Rendel, Bernhard Rumpe, Ina Schaefer, and Frank van der Linden. Hierarchical variability modeling for software architectures. In Eduardo Santana de Almeida, Tomoji Kishi, Christa Schwanninger, Isabel John, and Klaus Schmid, editors, *Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22-26, 2011*, pages 150–159. IEEE, 2011. *Cité page 67*

[HS11] Brian Henderson-Sellers. Random thoughts on multi-level conceptual modelling. In Roland Kaschek and Lois Delcambre, editors, *The evolution of conceptual modeling*, chapter Random thoughts on multi-level conceptual modelling, pages 93–116. Springer-Verlag, Berlin, Heidelberg, 2011. *Cité page 23*

[HSG11] Regina Hebig, Andreas Seibel, and Holger Giese. On the unification of megamodels. In Vasco Amaral, Hans Vangheluwe, Cécile Hardebolle, Laszlo Lengyel, Tiziana Magaria, Julia Padberg, and Gabriele Taentzer, editors, *Proceedings of the 4th International Workshop on Multi-Paradigm Modeling (MPM 2010)*, volume 42 of *Electronic Communications of the EASST*, 2011. *Cité page 24*

[HSS+10] Florian Heidenreich, Pablo Sánchez, João Santos, Steffen Zschaler, Mauricio Alférez, João Araújo, Lidia Fuentes, Uirá Kulesza, Ana Moreira, and Awais Rashid. Relating feature models to other models of a software product line: a comparative study of featuremapper and vml. In Shmuel Katz and Mira Mezini, editors, *Transactions on Aspect-Oriented Software Development VII: A Common Case Study for Aspect-Oriented Modeling*, number vol. 7 in Lecture Notes in Computer Science, chapter Relating feature models to other models of a software product line: a comparative study of featuremapper and VML, pages 69–114. Springer-Verlag, Berlin, Heidelberg, 2010. *3 citations pages 40, 41, et 53*

[HT08] Herman Hartmann and Tim Trew. Using feature diagrams with context variability to model multiple product lines for software supply chains. In *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC '08, pages 12–21, Washington, DC, USA, 2008. IEEE Computer Society. *5 citations pages 49, 53, 57, 84, et 210*

[HTM09] Herman Hartmann, Tim Trew, and Aart Matsinger. Supplier independent feature modelling. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 191–200, Pittsburgh, PA, USA, 2009. Carnegie Mellon University. *3 citations pages 48, 49, et 57*

[Hub12] Arnaud Hubaux. *Feature-based Configuration: Collaborative, Dependable, and Controlled*. PhD thesis, Université de Namur, 2012. *Cité page 48*

[IBK11] Paul Istoan, Nicolas Biri, and Jacques Klein. Issues in model-driven behavioural product derivation. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 69–78, New York, NY, USA, 2011. ACM. *2 citations pages 68 et 187*

[IEE00] IEEE. Ieee recommended practice for architectural description of software-intensive systems. *IEEE Std 1471-2000*, pages i–23, 2000. *Cité page 21*

[IKPJ11] Paul Istoan, Jacques Klein, Gilles Perrouin, and Jean-Marc Jézéquel. A metamodel-based classification of variability modeling approaches. In *In VARY International Workshop affiliated with ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems (Vary@MODELS2011)*, New Zeeland, october 2011. *2 citations pages 38 et 39*

[ISO06] ISO/IEC/IEEE. International standard - iso/iec 14764 ieee std 14764-2006 software engineering – software life cycle processes – maintenance. *ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998)*, 1:1–44, 2006. *Cité page 70*

[ISO11] ISO/IEC/IEEE. Systems and software engineering – architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1 –46, 1 2011. *Cité page 20*

[JABK08] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008. *Cité page 28*

[Jac95] Michael Jackson. *Software requirements & specifications: a lexicon of practice, principles and prejudices*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995. *Cité page 38*

[Jac02] Michael Jackson. Some basic tenets of description. *Software and Systems Modeling*, 1:5–9, 2002. *2 citations pages 16 et 203*

[Jaz02] Mehdi Jazayeri. On architectural stability and evolution. In Johann Blieberger and Alfred Strohmeier, editors, *Reliable Software Technologies - Ada-Europe 2002, 7th Ada-Europe International Conference on Reliable Software Technologies, Vienna, Austria, June 17-21, 2002, Proceedings*, volume 2361 of *Lecture Notes in Computer Science*, pages 13–23. Springer, 2002. *Cité page 67*

[JB04a] Michel Jaring and Jan Bosch. A taxonomy and hierarchy of variability dependencies in software product family engineering. In *Proceedings of the 28th Annual International Computer Software and Applications Conference - Volume 01*, COMPSAC '04, pages 356–361, Washington, DC, USA, 2004. IEEE Computer Society. *4 citations pages 52, 57, 59, et 210*

[JB04b] Michel Jaring and Jan Bosch. Variability dependencies in product family engineering. In Frank van der Linden, editor, *Software Product-Family Engineering*, volume 3014 of *Lecture Notes in Computer Science*, pages 81–97. Springer Berlin / Heidelberg, 2004. *4 citations pages 52, 57, 59, et 210*

[JB05] Anton Jansen and Jan Bosch. Software architecture as a set of architectural design decisions. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, WICSA '05, pages 109–120, Washington, DC, USA, 2005. IEEE Computer Society. *Cité page 214*

[JB10] Etienne Juliot and Jérôme Benois. Viewpoints creation using obeo designer or how to build eclipse dsm without being an expert developer ?, 2010. *2 citations pages 29 et 188*

[Jéz99] Jean-Marc Jézéquel. Reifying variants in configuration management. *ACM Trans. Softw. Eng. Methodol.*, 8(3):284–295, July 1999. *Cité page 41*

[JGB06] Jean-Marc Jézéquel, Sébastien Gérard, and Benoit Baudry. Le génie logiciel et l'idm : une approche unificatrice par les modèles. In J. Estublier, editor, *L'ingénierie dirigée par les modèles*. Lavoisier, Hermes-science, 2006. *Cité page 21*

[JGJ97] Ivar Jacobson, M. L. Griss, and P. Jonsson. *Reuse-driven Software Engineering Business (RSEB)*. Addison-Wesley, 1997. *2 citations pages 67 et 277*

[JLL99] Catherine Blake Jaktman, John Leaney, and Ming Liu. Structural analysis of the software architecture - a maintenance assessment case study. In *Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, WICSA1, pages 455–470, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V. *Cité page 70*

[Jou05] F. Jouault. Loosely coupled traceability for atl. *Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability, Nuremberg, Germany*, pages 29–37, 2005. *Cité page 31*

[Jør06] Kaj A. Jørgensen. Product modeling on multiple abstraction levels. In *Mass Customization: Challenges and Solutions*, volume 87 of *Int. Series in Operations Research & Management Science*, pages 63–84. Springer US, 2006. *Cité page 62*

[JV10] Isabel John and Karina Villela. Evolutionary product line scoping. In *Proceedings of the 14th international conference on Software product lines: going beyond*, SPLC'10, pages 515–516, Berlin, Heidelberg, 2010. Springer-Verlag. *Cité page 146*

[JZ05] Waraporn Jirapanthong and Andrea Zisman. Supporting product line development through traceability. In *Proceedings of the 12th Asia-Pacific Software Engineering Conference*, APSEC '05, pages 506–514, Washington, DC, USA, 2005. IEEE Computer Society. *Cité page 55*

[Jéz08] Jean-Marc Jézéquel. Model driven design and aspect weaving. *Journal of Software and Systems Modeling (SoSyM)*, 7(2):209–218, may 2008. *Cité page 20*

[JZ09] Waraporn Jirapanthong and Andrea Zisman. Xtraque: traceability for product line systems. *Software and Systems Modeling*, 8:117–144, 2009. *Cité page 55*

[Kai93] Hermann Kaindl. The missing link in requirements engineering. *SIGSOFT Softw. Eng. Notes*, 18(2):30–39, April 1993. *Cité page 29*

[Kazar] Rick Kazman. *Handbook of Software Engineering & Knowledge Engineering: Fundamentals*, volume 1 of *Handbook of Software Engineering & Knowledge Engineering*, chapter Software architecture. World Scientific Pub Co Inc, Decembre Year. *Cité page 214*

[KBA02] Ivan Kurtev, Jean Bézivin, and Mehmet Aksit. Technological spaces: An initial appraisal. In *CoopIS, DOA'2002 Federated Conferences, Industrial track*, 2002. *Cité page 23*

[KBK06] Rick Kazman, Len Bass, and Mark Klein. The essential components of software architecture design and analysis. *J. Syst. Softw.*, 79(8):1207–1216, August 2006. *Cité page 75*

[KC10] Ali Koudri and Joel Champeau. Modal: A spem extension to improve co-design process models. In Jürgen Münch, Ye Yang, and Wilhelm Schäfer, editors, *New Modeling Concepts for Today's Software Processes*, volume 6195 of *Lecture Notes in Computer Science*, pages 248–259. Springer Berlin / Heidelberg, 2010. *4 citations pages 5, 33, 95, et 96*

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990. *13 citations pages 3, 40, 44, 45, 48, 50, 57, 65, 67, 68, 205, 209, et 276*

[KCL05] Soo Dong Kim, Soo Ho Chang, and Hyun Jung La. Traceability map: Foundations to automate for product line engineering. *Software Engineering Research, Management and Applications, ACIS International Conference on*, 0:340–347, 2005. *Cité page 55*

[Ken02] Stuart Kent. Model driven engineering. In *Proceedings of the Third International Conference on Integrated Formal Methods*, IFM '02, pages 286–298, London, UK, UK, 2002. Springer-Verlag. *Cité page 13*

[KHC05] Soo Dong Kim, Jin Sun Her, and Soo Ho Chang. A theoretical foundation of variability in component-based development. *Inf. Softw. Technol.*, 47(10):663–673, July 2005. *Cité page 38*

[KKKL08] Taeho Kim, In-Young Ko, Sungwon Kang, and Danhyung Lee. Extending atam to assess product line architecture. In Qiang Wu, Xiangjian He, Quang Vinh Nguyen, Wenjing Jia, and Mao Lin Huang, editors, *Proceedings of 8th IEEE International Conference on Computer and Information Technology, CIT 2008, Sydney, Australia, July 8-11, 2008*, pages 790–797. IEEE, 2008. *Cité page 74*

[KKL+98] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, January 1998. *13 citations pages 40, 44, 45, 48, 50, 52, 57, 65, 67, 68, 205, 209, et 276*

[KLD02] K.C. Kang, Jaejoon Lee, and P. Donohoe. Feature-oriented product line engineering. *Software, IEEE*, 19(4):58 – 65, jul/aug 2002. *2 citations pages 65 et 276*

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Aksit and Satoshi Matsuoka, editors, *ECOOP'97 - Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Berlin / Heidelberg, 1997. *Cité page 20*

[KLV05] Paul Klint, Ralf Lämmel, and Chris Verhoef. Toward an engineering discipline for grammarware. *ACM Trans. Softw. Eng. Methodol.*, 14(3):331–380, July 2005. *Cité page 23*

[KPKH02] Antje von Knethen, Barbara Paech, Friedemann Kiedaisch, and Frank Houdek. Systematic requirements recycling through abstraction and traceability. In *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, RE '02, pages 273–281, Washington, DC, USA, 2002. IEEE Computer Society. *Cité page 31*

[Kru95] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12:42–50, 1995. *Cité page 21*

[Kru02] Charles W. Krueger. Easing the transition to software mass customization. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 282–293, London, UK, 2002. Springer-Verlag. *2 citations pages 68 et 72*

[Kru03] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003. *Cité page 214*

[Kru06] CharlesW. Krueger. New methods in software product line development. In *Proceedings of the 10th International on Software Product Line Conference*, SPLC '06, pages 95–102, Washington, DC, USA, 2006. IEEE Computer Society. *Cité page 73*

[Kru07] Charles W. Krueger. Biglever software gears and the 3-tiered spl methodology. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, OOPSLA '07, pages 844–845, New York, NY, USA, 2007. ACM. *3 citations pages 41, 45, et 68*

[KSG06] Michael Kircher, Christa Schwanninger, and Iris Groher. Transitioning to a software product family approach - challenges and best practices. In *Proceedings of the 10th International on Software Product Line Conference*, SPLC '06, pages 163–171, Washington, DC, USA, 2006. IEEE Computer Society. *Cité page 38*

[KSP09] K.C. Kang, V. Sugumaran, and S. Park. *Applied Software Product-Line Engineering*. Taylor & Francis, 2009. *3 citations pages 47, 62, et 271*

[KTS+09] Christian Kastner, Thomas Thum, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. Featureide: A tool framework for feature-oriented software development. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 611–614, Washington, DC, USA, 2009. IEEE Computer Society. *Cité page 45*

[Küh06] Thomas Kühne. Matters of (meta-)modeling. *Software and System Modeling*, 5(4):369–385, 2006. *4 citations pages 16, 22, 26, et 204*

[KWB03] A.G. Kleppe, J.B. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture : Practice and Promise*. The Addison-Wesley Object Technology Series. Addison-Wesley, 2003. *2 citations pages 19 et 28*

[KZ02] Alexander Kozlenkov and Andrea Zisman. Are their design specifications consistent with our requirements ? In *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, RE '02, pages 145–156, Washington, DC, USA, 2002. IEEE Computer Society. *Cité page 31*

[LBT09] Kwanwoo Lee, Goetz Botterweck, and Steffen Thiel. Aspectual separation of feature dependencies for flexible feature composition. In *Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference - Volume 01*, COMPSAC '09, pages 45–52, Washington, DC, USA, 2009. IEEE Computer Society. *Cité page 51*

[Leh80] M.M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980. *5 citations pages 61, 69, 74, 75, et 78*

[Leo74] Aleksei N. Leontiev. The problem of activity in psychology. *Soviet Psychology*, 13(2):4–33, 1974. *Cité page 33*

[Let02] Patricio Letelier. A framework for requirements traceability in uml-based projects. In *In Proc. of 1st Intl. Workshop on Traceability in Emerging Forms of Softw. Eng*, pages 32–41, 2002. *2 citations pages 29 et 31*

[LGB08] Miguel A. Laguna and Bruno González-Baixauli. Product line requirements: Multi-paradigm variability models. In Carme Quer, Juan Pablo Carvallo, and Lyrene Fernandes da Silva, editors, *Anais do WER08 - Workshop em Engenharia de Requisitos, Barcelona, Catalonia, Spain, September 12-13, 2008*, 2008. *Cité page 40*

[LHET+10] Roberto E. Lopez-Herrejon, Alexander Egyed, Salvador Trujillo, Josune de Sosa, and Maider Azanza. Using incremental consistency management for conformance checking in feature-oriented model-driven engineering. In David Benavides, Don S. Batory, and Paul Grünbacher, editors, *VaMoS*, volume 37 of *ICB-Research Report*, pages 93–100. Universität Duisburg-Essen, 2010. *Cité page 74*

[Liv11] Steve Livengood. Issues in software product line evolution: complex changes in variability models. In *Proceedings of the 2nd International Workshop on Product Line Approaches in Software Engineering*, PLEASE '11, pages 6–9, New York, NY, USA, 2011. ACM. *Cité page 74*

[LJZ11] Luis C. Lamb, Waraporn Jirapanthong, and Andrea Zisman. Formalizing traceability relations for product lines. In *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '11, pages 42–45, New York, NY, USA, 2011. ACM. *4 citations pages 55, 57, 59, et 211*

[LK04] Kwanwoo Lee and Kyo C. Kang. Feature dependency analysis for product line component design. In Jan Bosch and Charles Krueger, editors, *Software Reuse: Methods, Techniques, and Tools*, volume 3107 of *Lecture Notes in Computer Science*, pages 69–85. Springer Berlin / Heidelberg, 2004. *5 citations pages 44, 51, 52, 57, et 210*

[LK10] Kwanwoo Lee and Kyo C. Kang. Usage context as key driver for feature selection. In *Proceedings of the 14th international conference on Software product lines: going beyond*, SPLC'10, pages 32–46, Berlin, Heidelberg, 2010. Springer-Verlag. *8 citations pages 48, 49, 54, 57, 59, 84, 186, et 210*

[LKL02] Kwanwoo Lee, Kyo Chul Kang, and Jaejoon Lee. Concepts and guidelines of feature modeling for product line software engineering. In *Proceedings of ICSR-7*, pages 62–77, London, UK, UK, 2002. Springer-Verlag. *Cité page 48*

[LL04] Axel van Lamsweerde and Emmanuel Letier. From object orientation to goal orientation: A paradigm shift for requirements engineering. In Martin Wirsing, Alexander Knapp, and Simonetta Balsamo, editors, *Radical Innovations of Software and Systems Engineering in the Future*, volume 2941 of *LNCS*, pages 153–166. Springer Berlin / Heidelberg, 2004. *3 citations pages 33, 143, et 144*

[LLT05] Jing (Janet) Liu, Robyn R. Lutz, and Jeffrey M. Thompson. Mapping concern space to software architecture: a connector-based approach. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, May 2005. *Cité page 54*

[LMB+01] Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. The generic modeling environment. In *In 9th InternationalWorkshop on Intelligent Signal Processing (WISP 2001)*, Budapest, Hungary, 2001. *Cité page 23*

[Loe09] Jon Loeliger. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development.* O'Reilly Media, Inc., 1st edition, 2009. *Cité page 72*

[LPT09] Kim Lauenroth, Klaus Pohl, and Simon Toehning. Model checking of domain artifacts in product line engineering. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 269–280, Washington, DC, USA, 2009. IEEE Computer Society. *Cité page 41*

[LR01] M. M. Lehman and J. F. Ramil. Evolution in software and related areas. In *Proceedings of the 4th International Workshop on Principles of Software Evolution*, IWPSE '01, pages 1–16, New York, NY, USA, 2001. ACM. *Cité page 3*

[LS96] Mikael Lindvall and Kristian Sandahl. Practical implications of traceability. *Softw. Pract. Exper.*, 26(10):1161–1180, October 1996. *2 citations pages 29 et 30*

[LSGF08] Neil Loughran, Pablo Sánchez, Alessandro Garcia, and Lidia Fuentes. Language support for managing variability in architectural models. In *Proceedings of the 7th international conference on Software composition*, SC'08, pages 36–51, Berlin, Heidelberg, 2008. Springer-Verlag. *Cité page 67*

[LSR07] Frank J. Van Der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering.* Springer, 2007. *Cité page 3*

[Lub88] M. D. Lubars. Software reuse: emerging technology. In *Software reuse: emerging technology*, chapter Wide-spectrum support for software reusability, pages 275–281. IEEE Computer Society Press, Los Alamitos, CA, USA, 1988. *Cité page 44*

[Lud03] Jochen Ludewig. Models in software engineering - an introduction. *Software and Systems Modeling*, 2:5–14, 2003. *2 citations pages 16 et 203*

[LYZZ06] Yuqin Lee, Chuanyao Yang, Chongxiang Zhu, and Wenyun Zhao. An approach to managing feature dependencies for product releasing in software product lines. In *Proceedings of the 9th international conference on Reuse of Off-the-Shelf Components*, ICSR'06, pages 127–141, Berlin, Heidelberg, 2006. Springer-Verlag. *Cité page 52*

[Mac01] Leszek A. Maciaszek. *Requirements analysis and system design: developing information systems with UML.* Addison-Wesley Longman Ltd., Essex, UK, UK, 2001. *2 citations pages 142 et 216*

[Mac02] Alessandro Maccari. Experiences in assessing product family software architecture for evolution. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 585–592, New York, NY, USA, 2002. ACM. *Cité page 74*

[Man02] Mike Mannion. Using first-order logic for product line model validation. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 176–187, London, UK, UK, 2002. Springer-Verlag. *3 citations pages 46, 48, et 134*

[Mat04] Mari Matinlassi. Comparison of software product line architecture design methods: Copa, fast, form, kobra and qada. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 127–136, Washington, DC, USA, 2004. IEEE Computer Society. *2 citations pages 65 et 277*

[MB89] John Moore and Sidney C. Bailin. The kaptur environment: An operations concept. Technical report, CTA Incorporated, Rockville, Maryland, USA, June 1989. *Cité page 44*

[MBKM08] Thilo Mende, Felix Beckwermert, Rainer Koschke, and Gerald Meier. Supporting the grow-and-prune model in software product lines evolution using clone detection. In *Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*, CSMR '08, pages 163–172, Washington, DC, USA, 2008. IEEE Computer Society. *Cité page 74*

[McI68] M. D. McIlroy. Mass-produced software components. *Proc. NATO Conf. on Software Engineering, Garmisch, Germany*, 1968. *2 citations pages 35 et 214*

[MCNY07] Mikyeong Moon, Heung Seok Chae, Taewoo Nam, and Keunhyuk Yeom. A metamodeling approach to tracing variability between requirements and architecture in software product lines. In *Proceedings of the 7th IEEE International Conference on Computer and Information Technology*, CIT '07, pages 927–933, Washington, DC, USA, 2007. IEEE Computer Society. *Cité page 55*

[Men10] Tom Mens. Model transformation: A survey of the state-of-the-art. In Sebastien Gerard, Jean-Philippe Babau, and Joel Champeau, editors, *Model Driven Engineering for Distributed Real-Time Embedded Systems*. Wiley - ISTE, 2010. *Cité page 28*

[MFB09] Pierre-Alain Muller, Frédéric Fondement, and Benoit Baudry. Modeling modeling. In Andy Schürr and Bran Selic, editors, *Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Denver, CO, USA, October 4-9, 2009. Proceedings*, volume 5795 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2009. *Cité page 95*

[MFBC10] Pierre-Alain Muller, Frédéric Fondement, Benoît Baudry, and Benoît Combemale. Modeling modeling modeling. *Software and Systems Modeling*, 11:347–359, 2010. *17 citations pages 5, 17, 25, 26, 27, 33, 79, 80, 91, 95, 96, 98, 100, 209, 213, 271, et 273*

[MKMG97] Robert T. Monroe, Andrew Kompanek, Ralph Melton, and David Garlan. Architectural styles, design patterns, and objects. *IEEE Softw.*, 14(1):43–52, January 1997. *Cité page 67*

[MMMN03] Jonathan I. Maletic, Ethan V. Munson, Andrian Marcus, and Tien N. Nguyen. Using a hypertext model for traceability link conformance analysis. In *In Proc. of the 2nd Int. Workshop on Traceability in Emerging Forms of Software Engineering*, pages 47–54, 2003. *Cité page 31*

[MO04] Mira Mezini and Klaus Ostermann. Variability management with feature-oriented programming and aspects. In *Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, SIGSOFT '04/FSE-12, pages 127–136, New York, NY, USA, 2004. ACM. *Cité page 41*

[Moh03] K. Mohan. *Managing variability in software product families: A traceability-based approach.* Ph.d. dissertation, Georgia State University, Atlanta, GA, 2003. Department of Computer Information Systems. *Cité page 55*

[Mon08] Martin Monperrus. *La mesure des modèles par les modèles : une approche générative.* These, Université Rennes 1, October 2008. *Cité page 17*

[MoTAIL65] M.L. Minsky and Massachusetts Institute of Technology. Artificial Intelligence Laboratory. *Matter, Mind and Models.* AI memo. Massachusetts Institute of Technology, Project MAC, 1965. *2 citations pages 16 et 203*

[MP92] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems - specification.* Springer, 1992. *Cité page 78*

[MPFM08] Shahid Mujtaba, Kai Petersen, Robert Feldt, and Michael Mattsson. Software product line variability : A systematic mapping study. *Engineering*, 2008. *Cité page 38*

[MPH+07] Andreas Metzger, Klaus Pohl, Patrick Heymans, Pierre-Yves Schobbens, and Germain Saval. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. *Requirements Engineering Conference*, 1:243–253, 2007. *12 citations pages 4, 38, 40, 41, 42, 48, 53, 57, 59, 84, 187, et 210*

[MPL+09] Brice Morin, Gilles Perrouin, Philippe Lahire, Olivier Barais, Gilles Vanwormhoudt, and Jean-Marc Jézéquel. Weaving variability into domain metamodels. In Andy Schürr and Bran Selic, editors, *Proceedings of MODELS'09 Denver, CO, USA*, pages 690–705, Berlin Heidelberg, 2009. Springer Verlag. *5 citations pages 40, 54, 57, 69, et 211*

[MRA05] Ana Moreira, Awais Rashid, and Joao Araujo. Multi-dimensional separation of concerns in requirements engineering. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, RE '05, pages 285–296, Washington, DC, USA, 2005. IEEE Computer Society. *Cité page 38*

[MSD+12] Raúl Mazo, Camille Salinesi, Daniel Diaz, Olfa Djebbi, and Alberto Lora-Michiels. Constraints: The heart of domain and application engineering in the product lines engineering strategy. *IJISMD*, 3(2):33–68, 2012. *4 citations pages 50, 55, 57, et 211*

[MSG96] Randall R. Macala, Lynn D. Stuckey, Jr., and David C. Gross. Managing domain-specific, product-line development. *IEEE Softw.*, 13(3):57–67, May 1996. *Cité page 74*

[MT00] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.*, 26(1):70–93, January 2000. *Cité page 67*

[Mul06] Pierre-Alain Muller. *De la modélisation objet des logiciels à la metamodélisation des langages informatiques*. Hdr, Université Rennes 1, November 2006. *Cité page 23*

[MVG06] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electron. Notes Theor. Comput. Sci.*, 152:125–142, March 2006. *Cité page 28*

[MVL+08] Brice Morin, Gilles Vanwormhoudt, Philippe Lahire, Alban Gaignard, Olivier Barais, and Jean-Marc Jézéquel. Managing variability complexity in aspect-oriented modeling. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, MoDELS '08, pages 797–812, Berlin, Heidelberg, 2008. Springer-Verlag. *2 citations pages 41 et 62*

[MWC09] Marcilio Mendonca, Andrzej Wasowski, and Krzysztof Czarnecki. Sat-based analysis of feature models is easy. In *Proceedings of SPLC '09*, pages 231–240, Pittsburgh, PA, USA, 2009. Carnegie Mellon University. *Cité page 134*

[MWCC08] Marcilio Mendonca, Andrzej Wasowski, Krzysztof Czarnecki, and Donald Cowan. Efficient compilation techniques for large scale feature models. In *Proceedings of the 7th international conference on Generative programming and component engineering*, GPCE '08, pages 13–22, New York, NY, USA, 2008. ACM. *Cité page 134*

[MZZ06] Hong Mei, Wei Zhang, and Haiyan Zhao. A metamodel for modeling system features and their refinement, constraint and interaction relationships. *Software and Systems Modeling*, 5:172–186, 2006. *4 citations pages 51, 57, 59, et 210*

[NAB11a] Elisa Yumi Nakagawa, Pablo Oliveira Antonino, and Martin Becker. Exploring the use of reference architectures in the development of product line artifacts. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, pages 28:1–28:8, New York, NY, USA, 2011. ACM. *Cité page 67*

[NAB11b] Elisa Yumi Nakagawa, Pablo Oliveira Antonino, and Martin Becker. Reference architecture and product line architecture: a subtle but critical difference. In *Proceedings of the 5th European conference on Software architecture*, ECSA'11, pages 207–211, Berlin, Heidelberg, 2011. Springer-Verlag. *Cité page 67*

[NEFE03] Christian Nentwich, Wolfgang Emmerich, Anthony Finkelstein, and Ernst Ellmer. Flexible consistency checking. *ACM Trans. Softw. Eng. Methodol.*, 12(1):28–63, January 2003. *Cité page 21*

[Nei86] J. Neighbors. Readings in artificial intelligence and software engineering. In Charles Rich and Richard C. Waters, editors, *Readings in artificial intelligence and software engineering*, chapter The Draco approach to constructing software from reusable components, pages 525–535. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1986. *2 citations pages 44 et 65*

[NOQ+07] Daren Nestor, Luke O'Malley, Aaron J. Quigley, Ernst Sikora, and Steffen Thiel. Visualisation of variability in software product line engineering. In Klaus Pohl, Patrick Heymans, Kyo Chul Kang, and Andreas Metzger, editors, *VaMoS*, volume 2007-01 of *Lero Technical Report*, pages 71–78, 2007. *Cité page 49*

[NTB+08] Daren Nestor, Steffen Thiel, Goetz Botterweck, Ciarán Cawley, and Patrick Healy. Applying visualisation techniques in software product lines. In *Proceedings of the 4th ACM symposium on Software visualization*, SoftVis '08, pages 175–184, New York, NY, USA, 2008. ACM. *2 citations pages 45 et 49*

[oEE90] The Institute of Electrical and Eletronics Engineers. Ieee standard glossary of software engineering terminology. IEEE Standard, September 1990. *3 citations pages 30, 45, et 205*

[OM07] Femi G. Olumofin and Vojislav B. Mišić. A holistic architecture assessment method for software product lines. *Inf. Softw. Technol.*, 49(4):309–323, April 2007. *Cité page 74*

[OMG01] OMG. Model driven architecture (mda). Technical Report ormsc/01-07-01, Object Management Group, July 2001. *3 citations pages 18, 19, et 271*

[OMG03b] OMG. Mda guide v1.0.1. Technical Report omg/03-06-01, Object Management Group, June 2003. *4 citations pages 16, 19, 32, et 203*

[OMG05] OMG. Uml 2.0 superstructure specification. Technical Report formal/05-07-04, Object Management Group, July 2005. *Cité page 16*

[OMG06] OMG. Corba component model specification. Technical Report formal/06-04-01, Object Management Group, April 2006. *3 citations pages 19, 144, et 275*

[OMG07] OMG. Data distribution services specification. Technical Report formal/2007-01-01, Object Management Group, January 2007. *2 citations pages 144 et 275*

[OMG08] OMG. Software & systems process engineering metamodel specification (spem). Technical Report formal/2008-04-01, Object Management Group, April 2008. *2 citations pages 33 et 276*

[OMG11a] OMG. Meta object facility (mof) specification. Technical Report formal/2011-08-07, Object Management Group, August 2011. *6 citations pages 19, 22, 23, 24, 271, et 276*

[OMG11b] OMG. Mof/xmi specification. Technical Report formal/2011-08-09, Object Management Group, August 2011. *2 citations pages 19 et 276*

[OMG11c] OMG. Uml 2.0 infrastructure specification. Technical Report formal/2011-08-05, Object Management Group, August 2011. *4 citations pages 17, 18, 19, et 276*

[OMG11d] OMG. Uml 2.0 superstructure specification. Technical Report formal/2011-08-06, Object Management Group, August 2011. *4 citations pages 17, 18, 19, et 276*

[OMG12] OMG. Ocl v2.3.1 specification. Technical Report formal/2012-01-01, Object Management Group, January 2012. *2 citations pages 24 et 276*

[OMTR10] Pádraig O'Leary, Fergal McCaffery, Steffen Thiel, and Ita Richardson. An agile process model for product derivation in software product line engineering. *Journal of Software Maintenance and Evolution: Research and Practice*, 2010. *2 citations pages 68 et 69*

[OO07] Gøran K. Olsen and Jon Oldevik. Scenarios of traceability in model to text transformations. In *Proceedings of the ECMDA-FA'07*, pages 144–156. Springer-Verlag, 2007. *Cité page 30*

[OOK10] Jon Oldevik, Gøran K. Olsen, and Dimitrios Kolovos. Ecmda traceability workshop, 2005-2010. *Cité page 31*

[OS00] OMG and Richard Soley. Model driven architecture. White paper omg/00-11-05, Object Management Group, November 2000. *5 citations pages 14, 18, 19, 32, et 271*

[Par72] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, December 1972. *Cité page 214*

[Par76] D.L. Parnas. On the design and development of program families. *Software Engineering, IEEE Transactions on*, SE-2(1):1 – 9, march 1976. *Cité page 35*

[Par94] David Lorge Parnas. Software aging. In *Proceedings of the 16th international conference on Software engineering*, volume 352, pages 279–287, 1994. *2 citations pages 3 et 70*

[PBL05] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer NY, Inc., Secaucus, NJ, USA, 2005. *24 citations pages 2, 3, 4, 37, 38, 40, 41, 42, 43, 54, 55, 57, 59, 61, 62, 66, 67, 69, 84, 187, 205, 211, 271, et 2*

[PD91] Rubén Prieto-Díaz. Implementing faceted classification for software reuse. *Commun. ACM*, 34(5):88–97, May 1991. *Cité page 65*

[PD99] B.J. Pine and S. Davis. *Mass Customization: The New Frontier in Business Competition.* Harvard Business School Press, 1999. *Cité page 63*

[PG96] Francisco A. C. Pinheiro and Joseph A. Goguen. An object-oriented tool for tracing requirements. *IEEE Softw.*, 13(2):52–64, March 1996. *2 citations pages 29 et 30*

[Pig97] Thomas M. Pigoski. *Practical software Maintenance: best practices for managing your software investment.* John Wiley and Sons, 1997. *Cité page 61*

[Pil04] Michael Pilato. *Version Control With Subversion.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004. *Cité page 72*

[PK93] Sung Joo Park and Hyoung Do Kim. Constraint-based metaview approach for modeling environment generation. *Decis. Support Syst.*, 9(4):325–348, June 1993. *Cité page 29*

[PKGJ08] Gilles Perrouin, Jacques Klein, Nicolas Guelfi, and Jean-Marc Jézéquel. Reconciling automation and flexibility in product derivation. In *SPLC '08: 12th International Conference*, pages 339–348, Washington, DC, USA, 2008. IEEE Computer Society. *10 citations pages 4, 5, 40, 41, 43, 44, 62, 68, 69, et 76*

[Poh96a] Klaus Pohl. Pro-art: Enabling requirements pre-traceability. In *Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96)*, ICRE '96, pages 76–, Washington, DC, USA, 1996. IEEE Computer Society. *Cité page 29*

[Poh96b] Klaus Pohl. *Process-Centered Requirements Engineering*. John Wiley & Sons, Inc., New York, NY, USA, 1996. *3 citations pages 29, 30, et 31*

[POKZ08] Richard F. Paige, Gøran K. Olsen, Dimitrios S. Kolovos, and Steffen Zschaler. Building model-driven engineering traceability classifications. In *Proceedings of the ECMDA-Traceability Workshop (ECMDA-TW 08)*, pages 49–58, 2008. *2 citations pages 30 et 32*

[POON10] RichardF. Paige, GoranK. Olsen, Jon Oldevik, and Tor Neple, editors. *Special Issue on Traceability in Model-Driven Engineering*, volume 9 of *Software & Systems Modeling*. Springer-Verlag, september 2010. *Cité page 31*

[Pou96] Jeffrey S. Poulin. *Measuring software reuse: principles, practices, and economic models*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996. *Cité page 73*

[PRB11] Andreas Pleuss, Rick Rabiser, and Goetz Botterweck. Visualization techniques for application in interactive product configuration. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, pages 22:1–22:8, New York, NY, USA, 2011. ACM. *Cité page 190*

[Pre97] Christian Prehofer. Feature-oriented programming: A fresh look at objects. In Mehmet Aksit and Satoshi Matsuoka, editors, *ECOOP'97 - Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 419–443. Springer Berlin / Heidelberg, 1997. *Cité page 20*

[PVM+12] Gilles Perrouin, Gilles Vanwormhoudt, Brice Morin, Philippe Lahire, Olivier Barais, and Jean-Marc Jézéquel. Weaving variability into domain metamodels. *Software and Systems Modeling*, 11:361–383, 2012. *Cité page 40*

[PW92] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52, October 1992. *4 citations pages 66, 69, 70, et 217*

[Rab95] P. Rabardel. *Les hommes et les technologies: approche cognitive des instruments contemporains*. Collection U.: Série Psychologie. A. Colin, 1995. *Cité page 33*

[RBSP02] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Extending feature diagrams with uml multiplicities. In *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, June 2002. *2 citations pages 40 et 45*

[RD92] Balasubramaniam Ramesh and Vasant Dhar. Supporting systems development by capturing deliberations during requirements engineering. *IEEE Trans. Softw. Eng.*, 18(6):498–510, June 1992. *Cité page 31*

[RE93] B. Ramesh and M. Edwards. Issues in the development of a requirements traceability model. In *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*, pages 256 –259, jan 1993. *Cité page 32*

[RFG+05] Raghu Reddy, Robert France, Sudipto Ghosh, Franck Fleurey, and Benoit Baudry. Model composition - a signature-based approach. In *Aspect Oriented Modeling (AOM) Workshop*, Montego Bay, Jamaica, October 2005. *5 citations pages 28, 158, 160, 161, et 188*

[rFHN06] Jean rémy Falleri, Marianne Huchard, and Clémentine Nebut. C.: Towards a traceability framework for model transformations in kermeta. In *In: ECMDA-TW Workshop*, 2006. *Cité page 31*

[Ric02] J. Richter. *Applied Microsoft.NET framework programming.* Pro-Developer. Microsoft Press, 2002. *Cité page 18*

[Rie03] Matthias Riebisch. Towards a more precise definition of feature models. In M. Riebisch, J. O. Coplien, and D. Streitferdt, editors, *In Proceedings of Modelling Variability for Object-Oriented Product Lines ECOOP Workshop*, pages 64–76, Darmstadt, Germany, 2003. *7 citations pages 45, 49, 51, 53, 57, 205, et 210*

[Rie04] M. Riebisch. Supporting evolutionary development by feature models and traceability links. In *Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the*, ECBS '04, pages 370 – 377, Washington, DC, USA, 2004. IEEE Computer Society. *3 citations pages 51, 57, et 210*

[RJ01] Balasubramaniam Ramesh and Matthias Jarke. Toward reference models for requirements traceability. *IEEE Trans. Softw. Eng.*, 27(1):58–93, January 2001. *3 citations pages 29, 30, et 31*

[Rol98] Colette Rolland. A comprehensive view of process engineering. *Lecture Notes in Computer Science*, 1413:1–24, 1998. *2 citations pages 33 et 95*

[ROR11] Rick Rabiser, Pádraig O'Leary, and Ita Richardson. Key activities for product derivation in software product lines. *J. Syst. Softw.*, 84(2):285–300, February 2011. *2 citations pages 68 et 76*

[RP01] Matthias Riebisch and Ilka Philippow. Evolution of product lines using traceability. In *In: Proceedings of Workshop on Engineering Complex Object-Oriented Systems for Evolution at OOPSLA 2001. (2001) Published online at: http://www.dsg.cs.tcd.ie/ecoose/oopsla2001/papers.shtml. Program Understanding - Issues, Tools, and Future Directions 15*, 2001. *2 citations pages 55 et 74*

[RPG07] Andreas Rummler, Christoph Pohl, and Birgit Grammel. Improving traceability in model-driven development of business applications. In *Proceedings of 3rd ECMDA Traceability Workshop*, pages 7–15, 2007. *Cité page 30*

[RRR11] A. Rashid, J.C. Royer, and A. Rummler. *Aspect-Oriented, Model-Driven Software Product Lines: The Ample Way.* Cambridge University Press, 2011. *Cité page 41*

[RW06] Mark-Oliver Reiser and Matthias Weber. Managing highly complex product families with multi-level feature trees. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 146–155, USA, 2006. *2 citations pages 4 et 49*

[RW07] Mark-Oliver Reiser and Matthias Weber. Multi-level feature trees. *Requirements Engineering*, 12:57–75, 2007. *5 citations pages 48, 49, 53, 57, et 210*

[SB99] Mikael Svahnberg and Jan Bosch. Evolution in software product lines: two cases. *Journal of Software Maintenance*, 11(6):391–422, 1999. *4 citations pages 71, 217, 218, et 273*

[SBPM09] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009. *2 citations pages 23 et 29*

[SC97] Mary Shaw and Paul C. Clements. A field guide to boxology: Preliminary classification of architectural styles for software systems. In *Proceedings of the 21st International Computer Software and Applications Conference*, COMPSAC '97, pages 6–13, Washington, DC, USA, 1997. IEEE Computer Society. *Cité page 217*

[SCFC09] Jean-Sebastien Sottet, Gaëlle Calvary, Jean-Marie Favre, and Joëlle Coutaz. *Megamodeling and Metamodel-Driven Engineering for Plastic User Interfaces: Mega-UI*, pages 173–200. Human-Computer Interaction Series. Springer, 2009. *2 citations pages 25 et 95*

[SCK+96] Mark Simos, Dick Creps, Carol Klingler, Larry Levine, and Dean Allemang. Software technology for adaptable reliable systems (stars) organization domain modeling (odm) guidebook version 2.0. Technical Report STARS-VC-A025/001/00, Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996. *2 citations pages 65 et 277*

[SD07] Marco Sinnema and Sybren Deelstra. Classifying variability modeling techniques. *Inf. Softw. Technol.*, 49(7):717–739, July 2007. *Cité page 38*

[SDH06] Marco Sinnema, Sybren Deelstra, and Piter Hoekstra. The covamof derivation process. In Maurizio Morisio, editor, *Reuse of Off-the-Shelf Components, 9th International Conference on Software Reuse, ICSR 2006, Turin, Italy, June 12-15, 2006, Proceedings*, volume 4039 of *Lecture Notes in Computer Science*, pages 101–114. Springer, 2006. *Cité page 40*

[SDNB04] Marco Sinnema, Sybren Deelstra, Jos Nijhuis, and Jan Bosch. Covamof: A framework for modeling variability in software product families. In Robert L. Nord, editor, *Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings*, volume 3154 of *Lecture Notes in Computer Science*, pages 197–213. Springer, 2004. *4 citations pages 40, 54, 57, et 211*

[SDNB06] Marco Sinnema, Sybren Deelstra, Jos Nijhuis, and Jan Bosch. Modeling dependencies in product families with covamof. In *13th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS 2006), 27-30 March 2006, Potsdam, Germany*, pages 299–307. IEEE Computer Society, 2006. *Cité page 40*

[SE02] Daniel Simon and Thomas Eisenbarth. Evolutionary introduction of software product lines. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 272–282, London, UK, UK, 2002. Springer-Verlag. *Cité page 72*

[SE05] Mehrdad Sabetzadeh and Steve Easterbrook. Traceability in viewpoint merging: a model management perspective. In *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, TEFSE '05, pages 44–49, New York, NY, USA, 2005. ACM. *Cité page 32*

[SE07] Klaus Schmid and Holger Eichelberger. A requirements-based taxonomy of software product line evolution. *ECEASST*, 8, 2007. *Cité page 71*

[Sei03] Ed Seidewitz. What models mean. *IEEE Softw.*, 20:26–32, September 2003. *6 citations pages 16, 17, 22, 23, 26, et 203*

[Sel03] B. Selic. The pragmatics of model-driven development. *Software, IEEE*, 20(5):19 – 25, sept.-oct. 2003. *3 citations pages 16, 17, et 203*

[SFR06] Roberto Silveira Silva Filho and David F. Redmiles. Towards the use of dependencies to manage variability in software product lines. In Paul Clements and Dirk Muthig, editors, *Proceedings of the Workshop held in conjunction with the 10th Software Product Line Conference (SPLC-2006)*, pages 4–15. Fraunhofer IESE, October 2006. *3 citations pages 52, 57, et 210*

[SG96] Mary Shaw and David Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. *3 citations pages 66, 214, et 215*

[SG10] Reinhard Stoiber and Martin Glinz. Supporting stepwise, incremental product derivation in product line requirements engineering. In David Benavides, Don S. Batory, and Paul Grünbacher, editors, *Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria, January 27-29, 2010. Proceedings*, volume 37 of *ICB-Research Report*, pages 77–84. Universität Duisburg-Essen, 2010. *Cité page 65*

[SHT06] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. Feature diagrams: A survey and a formal semantics. In *RE*, pages 136–145. IEEE Computer Society, 2006. *Cité page 45*

[SHTB07] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Generic semantics of feature diagrams. *Comput. Netw.*, 51(2):456–479, 2007. *3 citations pages 44, 45, et 46*

[SK01] Juha Savolainen and Juha Kuusela. Consistency management of product line requirements. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, RE '01, pages 40–, Washington, DC, USA, 2001. IEEE Computer Society. *Cité page 38*

[SK03] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, 20(5):42–45, September 2003. *Cité page 28*

[SLFG09] Pablo Sánchez, Neil Loughran, Lidia Fuentes, and Alessandro Garcia. Engineering languages for specifying product-derivation processes in software product lines. In Dragan Gašević, Ralf Lämmel, and Eric Wyk, editors, *Software*

Language Engineering, Lecture Notes in Computer Science, chapter Engineering Languages for Specifying Product-Derivation Processes in Software Product Lines, pages 188–207. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. *2 citations pages 68 et 69*

[SME09] Rick Salay, John Mylopoulos, and Steve Easterbrook. Using macromodels to manage collections of related models. In *Proceedings of the 21st International Conference on Advanced Information Systems Engineering*, CAiSE '09, pages 141–155, Berlin, Heidelberg, 2009. Springer-Verlag. *2 citations pages 24 et 204*

[SNG10] Andreas Seibel, Stefan Neumann, and Holger Giese. Dynamic hierarchical mega models: comprehensive traceability and its efficient maintenance. *Softw. Syst. Model.*, 9(4):493–528, September 2010. *2 citations pages 24 et 32*

[SR02] Johannes Sametinger and Matthias Riebisch. Evolution support by homogeneously documenting patterns, aspects and traces. In *ECSMR'02 Proceedings*, 2002. *3 citations pages 53, 57, et 210*

[SRG11] Klaus Schmid, Rick Rabiser, and Paul Grünbacher. A comparison of decision modeling approaches in product lines. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 119–126, New York, NY, USA, 2011. ACM. *Cité page 39*

[SRP03] D. Streitferdt, M. Riebisch, and I. Philippow. Details of formalized relations in feature models using ocl. In *Proceedings of 10th IEEE International Conference on Engineering of Computer–Based Systems (ECBS 2003), Huntsville, USA. IEEE Computer Society*, pages 45–54, 2003. *3 citations pages 51, 57, et 210*

[STB+04] Mirjam Steger, Christian Tischer, Birgit Boss, Andreas Müller, Oliver Pertler, Wolfgang Stolz, and Stefan Ferber. Introducing pla at bosch gasoline systems: Experiences and practices. In Robert Nord, editor, *Software Product Lines*, volume 3154 of *Lecture Notes in Computer Science*, pages 173–176. Springer Berlin / Heidelberg, 2004. *Cité page 49*

[Sua11] David Suarez. Modélisation de l'aspect dynamique de la variabilité dans les lignes de produits. Master's thesis, ENIB, 2011. *Cité page 187*

[SV02] Klaus Schmid and Martin Verlage. The economic impact of product line adoption and evolution. *IEEE Softw.*, 19(4):50–57, July 2002. *3 citations pages 72, 73, et 271*

[SvGB05] Mikael Svahnberg, Jilles van Gurp, and Jan Bosch. A taxonomy of variability realization techniques: Research articles. *Softw. Pract. Exper.*, 35:705–754, July 2005. *4 citations pages 37, 38, 41, et 205*

[Swa76] E. Burton Swanson. The dimensions of maintenance. In *Proceedings of the 2nd international conference on Software engineering*, ICSE '76, pages 492–497, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press. *3 citations pages 61, 70, et 71*

[SWPH09] Ina Schaefer, Alexandre Worret, and Arnd Poetzsch-Heffter. A model-based framework for automated product derivation. In *Proceedings of the Workshop in Model based Approaches for Product Line Engineering (MAPLE 2009)*, 2009. *Cité page 68*

[SZ04] George Spanoudakis and Andrea Zisman. Software traceability: A roadmap. In *Handbook of Software Engineering and Knowledge Engineering*, pages 395–428. World Scientific Publishing, 2004. *3 citations pages 29, 30, et 31*

[SZLT+10] Andreas Svendsen, Xiaorui Zhang, Roy Lind-Tviberg, Franck Fleurey, Øystein Haugen, Birger Møller-Pedersen, and Gøran K. Olsen. Developing a software product line for train control: A case study of cvl. In Jan Bosch and Jaejoon Lee, editors, *Software Product Lines: Going Beyond - 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13-17, 2010. Proceedings*, volume 6287 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2010. *Cité page 41*

[SZPMK04] George Spanoudakis, Andrea Zisman, Elena Pérez-Miñana, and Paul Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105 – 127, 2004. *2 citations pages 29 et 31*

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002. *2 citations pages 63 et 65*

[TAS07] Salvador Trujillo, Gentzane Aldekoa, and Goiuria Sagardui. Tracking the evolution of feature oriented product lines. In Xavier Franch, editor, *XII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2007), Zaragoza, Spain, September 11-14, 2007. Actas*, pages 355–360. Thomson Editorial, 2007. *Cité page 74*

[TBD07] Salvador Trujillo, Don Batory, and Oscar Diaz. Feature oriented model driven development: A case study for portlets. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 44–53, Washington, DC, USA, 2007. IEEE Computer Society. *Cité page 62*

[TBK09] Thomas Thum, Don Batory, and Christian Kastner. Reasoning about edits to feature models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 254–264, Washington, DC, USA, 2009. IEEE Computer Society. *Cité page 134*

[TH02] Steffen Thiel and Andreas Hein. Systematic integration of variability into product line architecture design. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 130–153, London, UK, UK, 2002. Springer-Verlag. *3 citations pages 54, 57, et 211*

[TH03a] Jeffrey M. Thompson and Mats P. E. Heimdahl. Structuring product family requirements for n-dimensional and hierarchical product lines. *Requirements Engineering*, 8:42–54, 2003. *Cité page 49*

[TH03b] Jean-Christophe Trigaux and Patrick Heymans. Modelling variability requirements in software product lines: a comparative survey. Technical report, Institut d'Informatique FUNDP, 2003. *Cité page 38*

[TKES11] Thomas Thum, Christian Kastner, Sebastian Erdweg, and Norbert Siegmund. Abstract features in feature modeling. In *Proceedings of the 2011 15th International Software Product Line Conference*, SPLC '11, pages 191–200, Washington, DC, USA, 2011. IEEE Computer Society. *Cité page 44*

[TM11] Cheng Thao and Ethan V. Munson. Flexible support for managing evolving software product lines. In *Proceedings of the 2nd International Workshop on Product Line Approaches in Software Engineering*, PLEASE '11, pages 60–64, New York, NY, USA, 2011. ACM. *Cité page 74*

[TMD09] Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, January 2009. *Cité page 67*

[TOHS99] Peri Tarr, Harold Ossher, William Harrison, and Stanley M. Sutton, Jr. N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the 21st international conference on Software engineering*, ICSE '99, pages 107–119, New York, NY, USA, 1999. ACM. *Cité page 20*

[TP08] Rasha Tawhid and Dorina C. Petriu. Towards automatic derivation of a product performance model from a uml software product line model. In *Proceedings of the 7th international workshop on Software and performance*, WOSP '08, pages 91–102, New York, NY, USA, 2008. ACM. *Cité page 69*

[TTBC+09] Thein Than Tun, Quentin Boucher, Andreas Classen, Arnaud Hubaux, and Patrick Heymans. Relating requirements and feature configurations: a systematic approach. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 201–210, Pittsburgh, PA, USA, 2009. Carnegie Mellon University. *7 citations pages 4, 38, 53, 57, 59, 84, et 187*

[VDG08] Karina Villela, Joerg Doerr, and Anne Gross. Proactively managing the evolution of embedded system requirements. In *Proceedings of the 2008 16th IEEE International Requirements Engineering Conference*, RE '08, pages 13–22, Washington, DC, USA, 2008. IEEE Computer Society. *Cité page 74*

[VDJ10] Karina Villela, Jörg Dörr, and Isabel John. Evaluation of a method for proactively managing the evolving scope of a software product line. In Roel Wieringa and Anne Persson, editors, *Requirements Engineering: Foundation for Software Quality*, volume 6182 of *Lecture Notes in Computer Science*, pages 113–127. Springer Berlin / Heidelberg, 2010. *Cité page 74*

[vDKV00] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35(6):26–36, June 2000. *Cité page 18*

[vdL02] F. van der Linden. Software product families in europe: the esaps cafe projects. *Software, IEEE*, 19(4):41 – 49, jul/aug 2002. *Cité page 63*

[VG07] Markus Voelter and Iris Groher. Product line implementation using aspect-oriented and model-driven software development. In *Proceedings of the 11th International Software Product Line Conference*, SPLC '07, pages 233–242, Washington, DC, USA, 2007. IEEE Computer Society. *2 citations pages 41 et 62*

[vGB02] Jilles van Gurp and Jan Bosch. Design erosion: problems and causes. *J. Syst. Softw.*, 61(2):105–119, March 2002. *Cité page 70*

[Voi08] Jean-Luc Voirin. Method & tools for constrained system architecting. In INCOSE, editor, *18th Annual International Symposium of the International Council on Systems Engineering, INCOSE 2008*, volume 2 of 5, page 15, Twente, June 2008. INCOSE. *8 citations pages 18, 20, 21, 106, 178, 188, 271, et 272*

[vOvdLKM00] Rob van Ommering, Frank van der Linden, Jeff Kramer, and Jeff Magee. The koala component model for consumer electronics software. *Computer*, 33(3):78–85, March 2000. *3 citations pages 54, 65, et 67*

[Vra04] Valentino Vranic. Reconciling feature modeling: A feature modeling metamodel. In Mathias Weske and Peter Liggesmeyer, editors, *Net.ObjectDays*, volume 3263 of *Lecture Notes in Computer Science*, pages 122–137. Springer, 2004. *4 citations pages 52, 57, 59, et 210*

[VSG11] Thomas Vogel, Andreas Seibel, and Holger Giese. The role of models and megamodels at runtime. In *Proceedings of the 2010 international conference on Models in software engineering*, MODELS'10, pages 224–238, Berlin, Heidelberg, 2011. Springer-Verlag. *Cité page 25*

[vZ02] Jay van Zyl. Product line architecture and the separation of concerns. In *Proceedings of the Second International Conference on Software Product Lines*, SPLC 2, pages 90–109, London, UK, UK, 2002. Springer-Verlag. *2 citations pages 67 et 139*

[Wir71] Niklaus Wirth. Program development by stepwise refinement. *Commun. ACM*, 14(4):221–227, April 1971. *Cité page 13*

[WJSA06] Staale Walderhaug, Ulrik Johansen, Erlend Stav, and Jan Aagedal. Towards a generic solution for traceability in mdd. *ECMDA Traceability Workshop*, 2006. *Cité page 30*

[WL99] D.M. Weiss and C.T.R. Lai. *Software product-line engineering: a family-based software development process.* Addison-Wesley, 1999. *8 citations pages 35, 36, 37, 62, 65, 72, 205, et 276*

[WSB+08] J. White, D.C. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz-Cortes. Automated diagnosis of product-line configuration errors in feature models. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 225–234, 2008. *Cité page 134*

[WvP10] Stefan Winkler and Jens von Pilgrim. A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling*, 9:529–565, 2010. *Cité page 31*

[yHMpOS04] Øystein Haugen, Birger Møller-pedersen, Jon Oldevik, and Arnor Solberg. An mda-based framework for model-driven product derivation. In *Software Engineering and Applications*, pages 709–714. ACTA Press, 2004. *Cité page 69*

[YL05] H. Ye and H. Liu. Approach to modelling feature variability and dependencies in software product lines. *Software, IEE Proceedings* -, 152(3):101 – 109, june 2005. *3 citations pages 52, 57, et 210*

[YM94] Eric S. K. Yu and John Mylopoulos. Understanding "why" in software process modelling, analysis, and design. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, pages 159–168, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. *3 citations pages 26, 33, et 95*

[Yu97] Eric S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, RE '97, pages 226–, Washington, DC, USA, 1997. IEEE Computer Society. *Cité page 33*

[ZHJ03] Tewfik Ziadi, Loïc Hélouët, and Jean-Marc Jézéquel. Towards a uml profile for software product lines. In Frank van der Linden, editor, *Software Product-Family Engineering, 5th International Workshop, PFE 2003, Siena, Italy, November 4-6, 2003, Revised Papers*, volume 3014 of *Lecture Notes in Computer Science*, pages 129–139. Springer, 2003. *3 citations pages 40, 69, et 178*

[ZJ97] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, January 1997. *Cité page 38*

[ZJ06] Tewfik Ziadi and Jean-Marc Jézéquel. Product line engineering with the uml: Deriving products. In K. Pohl, editor, *Software Product Lines*, pages 557–586. Springer Verlag, 2006. *4 citations pages 40, 67, 68, et 69*

[ZLZZ06] Chongxiang Zhu, Yuqin Lee, Wenyun Zhao, and Jingzhou Zhang. A feature oriented approach to mapping from domain requirements to product line architecture. In Hamid R. Arabnia and Hassan Reza, editors, *Proceedings of the International Conference on Software Engineering Research and Practice & Conference on Programming Languages and Compilers, SERP 2006, Las Vegas, Nevada, USA, June 26-29, 2006, Volume 1*, pages 219–225. CSREA Press, 2006. *4 citations pages 52, 57, 59, et 210*

[ZMZ05] Wei Zhang, Hong Mei, and Haiyan Zhao. A feature-oriented approach to modeling requirements dependencies. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, RE '05, pages 273–284, Washington, DC, USA, 2005. IEEE Computer Society. *Cité page 51*

[ZSP+11] Gang Zhang, Liwei Shen, Xin Peng, Zhenchang Xing, and Wenyun Zhao. Incremental and iterative reengineering towards software product line: An industrial case study. In *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance*, ICSM '11, pages 418–427, Washington, DC, USA, 2011. IEEE Computer Society. *Cité page 74*

[ZSPmK03] Andrea Zisman, George Spanoudakis, Elena Pérez-miñana, and Paul Krause. Tracing software requirements artefacts. In *In: The 2003 International Conference on Software Engineering Research and Practice (SERP'03). 2003. Las Vegas*, pages 448–455, 2003. *2 citations pages 30 et 31*

[ZSS+10] Steffen Zschaler, Pablo Sánchez, João Santos, Mauricio Alférez, Awais Rashid, Lidia Fuentes, Ana Moreira, João Araújo, and Uirá Kulesza. Vml* - a family of languages for variability management in software product lines. In Mark van den Brand, Dragan Gašević, and Jeff Gray, editors, *Software Language Engineering*, volume 5969 of *Lecture Notes in Computer Science*, pages 82–102. Springer Berlin / Heidelberg, 2010. *Cité page 40*

[ZZM04] Wei Zhang, Haiyan Zhao, and Hong Mei. A propositional logic-based method for verification of feature models. In Jim Davies, Wolfram Schulte, and Mike Barnett, editors, *Formal Methods and Software Engineering*, volume 3308 of

Lecture Notes in Computer Science, pages 115–130. Springer Berlin / Heidelberg, 2004. *Cité page 51*

[ZZM08] Haiyan Zhao, Wei Zhang, and Hong Mei. Multi-view based customization of feature models. *Journal of Frontier of Computer Science and Technology*, 2008. *3 citations pages 52, 57, et 210*

Sitographie

[1] ANRT. Cifre - conventions industrielles de formation par la recherche. http://www.anrt.asso.fr/fr/espace_cifre/accueil.jsp. réf. de janvier 2013. *Cité page 275*

[2] Eclipse. Eclipse modeling framework. http://www.eclipse.org/modeling/emf/. réf. de septembre 2012. *4 citations pages 23, 29, 188, et 276*

[3] Eclipse. Emf compare. www.eclipse.org/emf/compare/. réf. de octobre 2012. *Cité page 161*

[4] handbookofsoftwarearchitecture. Handbook of software architecture. http://www.handbookofsoftwarearchitecture.com. réf. de septembre 2012. *Cité page 140*

[5] IBM. Rational requisitepro. http://www-142.ibm.com/software/products/fr/fr/reqpro/. réf. de septembre 2012. *Cité page 31*

[6] IBM. Rational rhapsody. https://www.ibm.com/developerworks/rational/products/rhapsody/. réf. de juin 2013. *Cité page 188*

[7] IBM. Rational software architect. https://www.ibm.com/developerworks/rational/products/rsa/. réf. de juin 2013. *Cité page 188*

[8] ikv++ technologies ag. medini qvt. http://projects.ikv.de/qvt. réf. de septembre 2012. *Cité page 28*

[9] Software Engineering Institute. Software product lines. http://www.sei.cmu.edu/productlines/. réf. de septembre 2012. *4 citations pages 62, 65, 66, et 277*

[10] Software Engineering Institute. Software product lines conference. http://www.splc.net/. réf. de octobre 2012. *Cité page 277*

[11] Obeo. Obeo designer. http://www.obeodesigner.com/. réf. de septembre 2012. *2 citations pages 29 et 188*

[12] Obeo. Obeo traceability. http://obeo.fr/pages/obeo-traceability/fr. réf. de juin 2012. *Cité page 31*

[13] OMG. Mda success stories. http://www.omg.org/mda/products_success.htm. réf. de avril 2012. *Cité page 32*

[14] OMG. Object management group. http://www.omg.org/. réf. de janvier 2013. *Cité page 276*

[15] OMG. Semantic information modeling for federation rfp. http://www.omgwiki.org/architecture-ecosystem/doku.php?id=semantic_information_modeling_for_federation_rfp. réf. de septembre 2013. *Cité page 196*

[16] ORACLE. Java platform, entreprise edition (java ee). http://www.oracle.com/technetwork/java/javaee/overview/index.html. réf. de janvier 2013. *Cité page 18*

[17] Dassault Systemes. Reqtify. http://www.3ds.com/products/catia/portfolio/geensoft/geensoft-product-lines/reqtify/. réf. de septembre 2012. *Cité page 31*

[18] IRISA Triskell team. Kermeta. http://www.kermeta.org/. réf. de avril 2012. *2 citations pages 28 et 188*

Listes et tables

Liste des tableaux

Liste des listings

Table des figures

Acronymes

Terminologie générale

CIFRE : Conventions Industrielles de Formation par la Recherche [1]
CONOPS : CONcept of OPerationS
NFR : Non-Functional Requirement
OTAN : Organisation du traité de l'Atlantique Nord
ROI : Return On Investment
SdF : Sûreté de Fonctionnement
SUS : System Under Study

Terminologie spécifique Thales

ARCADIA : ARChitecture Analysis & Design Integrated Approach
EPBS : End-Product Breakdown Structure
IVVQ : intégration, validation, vérification, qualification

Terminologie Informatique

AOP : Aspect Oriented Programming
AST : Abstract Syntax Tree
BDD : Binary Decision Diagram
BNF : Backus Naur Form
CASE : Computer-Aided Software Engineering
EBNF : Extended Backus-Naur Form
FOP : Feature Oriented Programming
POO : Programmation Orientée Objet
SAT : boolean SATisfiability problem
XML : eXtensible Markup Language

Terminologie de l'ingénierie dirigée par les modèles

ATL : ATLAS Transformation Language
CCM : CORBA Component Model
CIM : Computational Independent Model
CORBA : Common Object Request Broker Architecture [OMG06]
DDS : Data Distribution Service [OMG07]
DSL : Domain-Specific Language

DSML : Domain Modeling Specific Language
ECore : EMF Core
EMF : Eclipse Modeling Framework [2]
EMOF : Essential MOF
GMF : Graphical Modeling Framework
IDM : Ingénierie Dirigée par les Modèles
MBE : Model-Based Engineering
MDA : Model-Driven Architecture
MDD : Model-Driven Development
MDE : Model-Driven Engineering
MM : MetaModel
MOF : Meta Object Facility [OMG11a]
OAM : Aspect Oriented Modeling
OCL : Object Constraint Language [OMG12]
OMG : Object Management Group [14]
PDM : Platform Description Model
PIM : Platform Independent Model
PSM : Platform Specific Model
QVT : Query/View/Transformation
SoC : Separation of Concerns
SPEM : Software & Systems Process Engineering Metamodel [OMG08]
UML : Unified Modeling Language [OMG11d, OMG11c]
XMI : XML Metadata Interchange [OMG11b]

Terminologie de l'architecture

ADL : Architecture Description Language
COTS : Components Off The Shelf
DEC : Derived Entity Control [Evi00]
GoF : Gang of Fours
MVC : Model-View-Controller
PAC : Presentation Abstraction Control [Cou87]
PLA : Product Line Architecture

Terminologie des lignes de produits

CVL : Common Variability Language [FHMP+09]
DA : Domain Analysis
DARE : Domain Analysis and Reuse Envirronment [FPDF98]
ESPLEP : Evolutionary Software Product Line Engineering Process [Gom04]
FAST : Family-oriented Abstraction Specification and Translation [WL99]
FeatuRSEB : Feature Reuse-Driven Software Engineering Business [GFA98]
FM : Feature model
FODA : Feature Oriented Domain Analysis [KCH+90]
FOPLE : Feature Oriented Product Line Software Engineering [KLD02]
FORM : Feature Oriented Reuse Method [KKL+98]
LdP : Ligne de Produits
LdPDM : Ligne de Produits Dirigée par les Modèles
MDSPL : Model Driven Software Product Line

ODM : Organization Domain Modeling [SCK+96].
OVM : Orthogonal Variability Model [PBL05]
PLA : Product Line Architecture
PLPI : SEI Product Line Practice Initiative [CN01, 9],
PLUSS : Product Line Use case modeling for Systems and Software engineering [EBB05],
PuLSE : ProdUct Line Software Engineering [BFK+99],
QADA : Quality-driven Architecture Design and quality Analysis [Mat04],
RSEB : Reuse-Driven Software Engineering Business [JGJ97]
SPL : Software Product Line
SPLC : Software Product Line Conference [10]
SPLE : Software Product Line Engineering
VP : Variation Point

Terminologie de l'approche

CFM : Context Feature Model
CVM : Context Variability Model
EFM : External Feature Model
EVM : External Variability Model
IFM : Internal Feature Model
IVM : Internal Variability Model
PLiMoS : Product Line Modeling Space language, langage de modélisation de l'espace de modélisation dédié aux LdPs

Anglicismes fréquemment employés

Goal-case : cas d'utilisation orienté but, cf. use-case
Use-case : cas d'utilisation d'UML

Glossaire

Analyse du domaine : processus de capture et de représentation de l'information relative aux applications dans un domaine donné, plus spécifiquement des caractéristiques communes et variables.

Approche proactive : approche de *ligne de produits* de production de masse, la proactivité se retrouve dans l'adoption et dans l'évolution. Elle est opposée à l'approche réactive, cf. section 4.4.3, page 72.

Architecture des Lignes de Produits : (en anglais *Product Line Architecture, PLA*) cf. définitions en section 4.2.3, page 66.

Architecture Logicielle : cf. définitions en section 4.2.1, page 66.

Configuration : processus de résolution des choix de variabilité et de validation de la cohérence de la solution obtenue. C'est une phase de l'ingénierie de l'application dans les approches de modélisation de variabilité dites par configuration (expression exhaustive des variabilités et commonalités puis restriction de ce domaine des possibles).

Core Assets : artefacts (logiciels) manipulés dans la production de produits dans un processus de LdP. Un *core asset* peut être de niveau architecture, composant logiciel, document, modèle, code, ou texte par exemple.

Dérivation : processus de réalisation, automatique ou manuelle, d'un produit à partir d'éléments réutilisables et du plan de production imposé par la *ligne de produits*.

Domaine : aire de connaissance ou d'activité caractérisée par un ensemble de concepts et une terminologie compréhensible des experts de ce domaine.

Élément variable : désignation générique d'un artefact de modélisation variable, suivant le type de modélisation, il peut être assimilé à un *feature*, une décision, un variant, etc.

Espace technique : cf. définition 4, page 24.

Famille de Produits : gamme de produits réalisés depuis d'une *ligne de produits*, cf. définition 13, page 64. La notion proche de catalogue de produits fait référence à un point de vue marketing.

Feature : élément variable du modèle du même nom, cf. définition 7, page 44.

Ingénierie de l'Application : (en anglais *Application Engineering*) processus d'ingénierie orienté vers le développement de produits à partir de solutions partielles ou connaissances encapsulées dans des modules.

Ingénierie du Domaine : (en anglais *Domain Engineering*) processus d'ingénierie orienté vers le développement pour la réutilisation, il comprend des activités d'analyse du domaine, de modélisation de la variabilité, d'élaboration de l'architecture de la LdP, et de réalisation de module d'encapsulation de connaissances *(core assets)*.

Ligne de Produits : (en anglais *Product Line*) cf. définition 11, page 63.

Méta-métamodèle : cf. définition 3, page 23.

Métamodèle : cf. définition 2, page 22.

Modèle : cf. définition 1, page 16.

Modèle de features : modèle de représentation de la variabilité à base de *features*, cf. section 3.3.3.

Modélisation par décisions : modélisation de la variabilité sans modélisation de la commonalité.

Patron d'architecture : motif de conception architectural d'un niveau d'abstraction élevé, cf. section B.3.1, page 215.

Périmètre de la Ligne de Produit : présence de la *ligne de produits* sur différents secteurs clés. Il est déterminé par la famille de produits réalisés, et se décline sur le plan du positionnement sur le marché, de la gamme de services et capacités offerts, et des possibilités de la plate-forme technique.

Point de variation : artefact d'identification de la variabilité et de sa résolution, définition 5, page 36.

Produit : extrant de la *ligne de produits*, cf. définition 12, page 64.

Variabilité : cf. définition 6, page 37.

Variant : représentation d'un artefact variable du domaine.

Une modélisation de la variabilité multidimensionnelle
pour une évolution incrémentale des lignes de produits

Résumé : Le doctorat s'inscrit dans le cadre d'une bourse CIFRE et d'un partenariat entre l'ENSTA Bretagne, l'IRISA et Thales Air Systems. Les préoccupations de ce dernier, et plus particulièrement de l'équipe de rattachement, sont de réaliser des systèmes à logiciels prépondérants embarqués. La complexité de ces systèmes et les besoins de compétitivité associés font emerger la notion de *Model-Based Product Lines (MBPLs)*. Celles-ci tendent à réaliser une synergie de l'abstraction de l'Ingénierie Dirigée par les Modèles (IDM) et de la capacité de gestion de la capitalisation et réutilisation des Lignes de Produits (LdPs). La nature irrévocablement dynamique des systèmes réels induit une évolution permanente des LdPs afin de répondre aux nouvelles exigences des clients et pour refléter les changements des artefacts internes de la LdP.

L'objectif de cette thèse est unique, maîtriser des incréments d'évolution d'une ligne de produits de systèmes complexes, les contributions pour y parvenir sont duales. La thèse est que 1^o) une variabilité multidimensionnelle ainsi qu'une modélisation relationnelle est requise dans le cadre de lignes de produits de systèmes complexes pour en améliorer la compréhension et en faciliter l'évolution (proposition d'un cadre générique de décomposition de la modélisation et d'un langage *(DSML)* nommé PLiMoS, dédié à l'expression relationnelle et intentionnelle dans les MBPLs), et que 2^o) les efforts de spécialisation lors de la dérivation d'un produit ainsi que l'évolution de la LdP doivent être guidé par une architecture conceptuelle (introduction de motifs architecturaux autour de PLiMoS et du patron ABCDE) et capitalisés dans un processus outillé semi-automatisé d'évolution incrémentale des lignes de produits par extension.

Mots-clés : IDM ; Ligne de Produits ; Espace de modélisation ; Modélisation relationnelle ; Variabilité multiple ; Modèles hétérogènes ; Architecture ; Évolution incrémentale ; Langage PLiMoS.

A multidimensionnal variability modeling
for an incremental product line evolution

Abstract: The PhD (CIFRE fundings) was supported by a partnership between three actors: ENSTA Bretagne, IRISA and Thales Air Systems. The latter's concerns, and more precisely the ones from the affiliation team, are to build embedded software-intensive systems. The complexity of these systems, combined to the need of competitivity, reveal the notion of Model-Based Product Lines (MBPLs). They make a synergy of the capabilities of modeling and product line approaches, and enable more efficient solutions for modularization with the distinction of abstraction levels and separation of concerns. Besides, the dynamic nature of real-world systems induces that product line models need to evolve continually to meet new customer requirements and to reflect changes in product line artifacts.

The aim of the thesis is to handle the increments of evolution of complex systems product lines, the contributions to achieve it are twofolds. The thesis claims that *i*) a multidimensional variability and a relational modeling are required within a complex system product line in order to enhance comprehension and ease the PL evolution (Conceptual model modularization framework and PLiMoS Domain Specific Modeling Language proposition ; the language is dedicated to relational and intentional expressions in MBPLs), and that *ii*) specialization efforts during product derivation have to be guided by a conceptual architecture (architectural patterns on top of PLiMoS, e.g. ABCDE) and capitalized within a semi-automatic tooled process allowing the incremental PL evolution by extension.

Keywords: MDE ; Product Line ; Modeling Space ; Relational Modeling ; Multiple Variability ; Heterogeneous Models ; Architecture ; Incremental Evolution ; PLiMoS Language.